

ISSN 0933-9434 | März/April 1989 | 3. Jg./Heft 2
ÖS 150,- | DM 19,80 | sfr 19,80

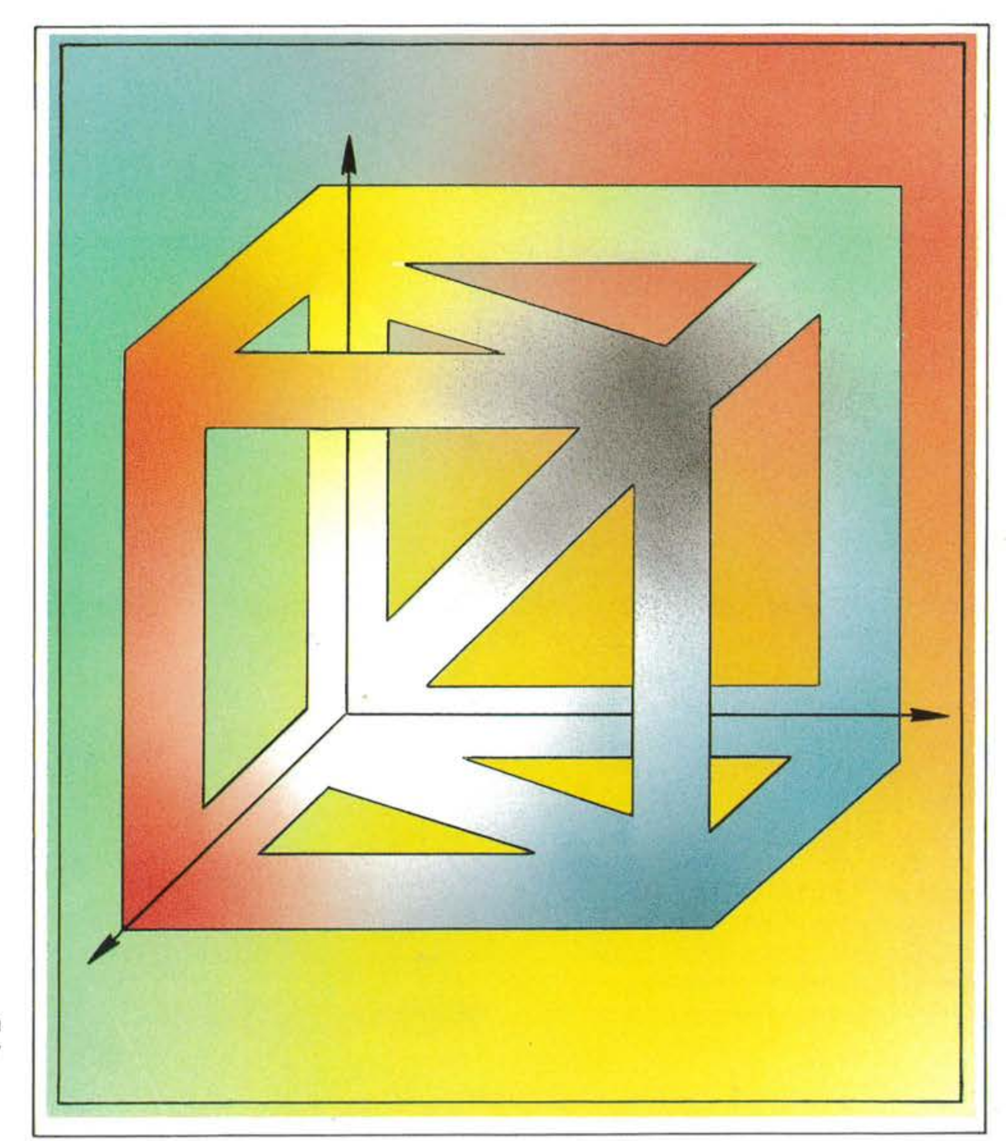
Windows:

Die Farbsysteme von Windows in Theorie und Praxis
Fenster-Unterklassen und ihre Verwendung
Eine nützliche Utility erweitert die Zwischenablage

PC-Know-how:

Das MS-Word-Dateiformat

64 Kbyte mehr Speicher unter DOS



SAA-Serie:

Maussteuerung und Tastaturabfrage

Sprachen:

Das neue QuickBASIC

OS/2 und DOS:

Eine Bibliothek bringt Presentation Manager-Programme unter DOS



DIALOG-MANAGER

Werkzeug zur Entwicklung graphischer Benutzeroberflächen

Komplexe Anwendungen mit graphischen Oberflächen werden sich in Zukunft am Markt immer mehr durchsetzen. Der Dialog-Manager ist ein Werkzeug, mit dem graphische Benutzeroberflächen in Fenstersystemen ohne Programmierkenntnisse erstellt und bearbeitet werden können. Die dazugehörige Anwendung wird mit Unterstützung durch den Dialog-Manager in einfacher Weise an die generierte Benutzerschnittstelle angebunden.

- **effiziente Entwicklung**
- **schnelle Überprüfung**
- **erhebliche Kosteneinsparung**
- **schnelle Dialogprogrammierung**
- **schnelles Prototyping**
- **einfache Wartung und Änderung**
- **leichte Portierbarkeit**

Der Dialog-Manager ist erhältlich für die Fenstersysteme:

- MS-Windows (ab Version 2.03)
- Presentation-Manager
- X Window System (ab Version 11.3 für Xr und Xt-Toolkit)
- αWindows (für Terminals)

unter den Betriebssystemen DOS, OS/2 und UNIX.



ISA Informationssysteme GmbH
Azenbergstraße 35, 7000 Stuttgart 1
Telefon (0711) 2 27 69-0
Fax (0711) 2 27 69-19
USEnet: info @ isaak.uucp

Inhalt

Ausgabe März/April 1989





Das Windows-Programm Scrapbook+:

Eine gelungene Erweiterung der Zwischenablage

Die Zwischenablage von Windows, das Clipboard, ist wahrscheinlich die meistverwendete Einrichtung zum Datenaustausch zwischen Windows-Anwendungen. Mit der Zwischenablage kann eine in einem Programm erstellte Kalkulationstabelle oder Zeichnung einfach in ein anderes Progamm übertragen werden, und zwar ohne dazu eine Zwischendatei anlegen zu müssen.

Windows enthält eine selten genutzte Anwendung, die Datei CLIPBRD.EXE zum Betrachten der Zwischenablage. Dieses Programm erlaubt Ihnen also, den momentanen Inhalt der Zwischablage anzusehen. Dummerweise besitzt es darüber hinaus kaum andere Fähigkeiten.

Wenn man die Windows-Zwischenablage und den angeschlossenen »Betrachter« eine Weile benutzt hat, bemerkt man folgende Einschränkungen:

- Die Zwischenablage ist auf eine Dateneinheit begrenzt (allerdings eventuell in mehreren Formaten).
- Der aktuelle Inhalt der Zwischenablage kann weder gelöscht noch gespeichert werden.
- Oft wird das erzeugende Programm gebraucht, um seine Daten in die Zwischenablage zu bringen.
- Es gibt kein vom System unterstütztes Hilfsmittel, um das Format der Zwischenablage zu manipulieren.
- Die Formate der Zwischenablagedaten können nicht beeinflußt werden.

Scrapbook+ von der amerikanischen Firma T/Maker ist eine anspruchsvolle Windows-Anwendung, die entwickelt wurde, um einige Einschränkungen der Zwischenablage zu überwinden. Da Scrapbook+ als Vermittler zu den Daten der Zwischenablage zur Verfügung steht, können Sie gelegentlich gebrauchte Daten in leicht erreichbaren Dateien ablegen. Wenn Sie die Daten dann brauchen, sind sie leicht durch Einkopieren (paste) aus der Zwischenablage in jedem Windows-Programm verfügbar. Durch die automatische Erzeugung kleiner Darstellungen des Inhalts in einem visuellen Index ermöglicht Scrapbook+, schnell seitenweise durch Abbildungen und Daten aller Arten zu blättern, und sie dann nahezu unverzüglich über die Zwischenablage in eine andere Windows-Anwendung zu übertragen. Scrapbook+ wurde nach der Methode entwickelt, die wir im Artikel »Datenaustausch unter Windows« im *Microsoft System Journal* 11/12 '88 beschrieben haben. *Bild 1* veranschaulicht die Auswahl einer Grafik in Scrapbook+ und *Bild 2* zeigt, wie dieses »Kunstwerk« in Windows Write »eingeklebt« wird.

Konzepte der Enwicklung

Hinter jedem Produkt stehen Anforderungen, Ergebnisse, Enttäuschungen und Träume - und Scrapbook+ bildet da auch keine Ausnahme. Die ursprüngliche Produktidee entstand im Frühjahr 1987 in einer Reihe informeller Gespräche zwischen uns und der Belegschaft von T/Marker, dem Herausgeber von Scrapbook+.

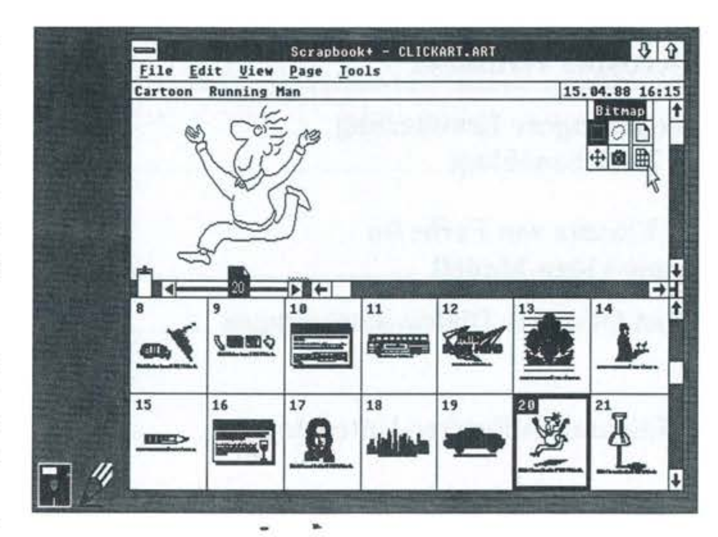


Bild 1: *Scrapbook+ bietet Anwendern zahlreiche Werkzeuge für die Auswahl und Bearbeitung von Bildern.*

Ursprünglich als Clip-Art-Manager geplant, entwickelte Scrapbook+ sich schnell zu einem umfangreichen Werkzeug zum Speichern und Abfragen von Zwischenablagedaten. Ganz am Anfang wollten wir ein einfaches Produkt entwickeln, das ebenso leicht zu erlernen wie anschaulich, schnell und verläßlich seine sollte. Noch wichtiger, wir suchten ein Programm zu schaffen, das die Benutzer für so unentbehrlich halten sollten wie etwa die Norton Utilities.

Ein Werkzeug zum Speichern von Daten

Eines der Hauptprobleme der Windows-Zwischenablage ist ihre flüchtige Natur. Wir wollten dem Benutzer ein Mittel an die Hand geben, mit dem er die Daten für späteren Gebrauch speichern kann. Anfangs hofften wir, jede Art von Daten speichern zu können, aber im Verlauf der Entwicklung von Scrapbook+ beschränkten wir uns auf allgemein genutzte und gut verständliche Formate.

Außer dem Import von Daten aus der Zwischenablage in Scrapbook+ hielten wir es für notwendig, gleiches auch direkt von der Diskettte bzw. Platte zu ermöglichen. Die Funktionalität würde wesentlich erweitert werden, wenn der Anwender noch andere Arten von Informationen im Zugriff hätte, von denen einige nicht einmal in Windows erreichbar wären. Obwohl wir erkannten, daß dieses Vorhaben die Entwicklung einer komplizierten Konvertierungs-Routine von einer Datei zur Zwischenablage erfordern würde, fühlten wir uns fast dazu verpflichtet, um ein vollständiges Produkt zu erhalten.

Nachdem wir die Datenkonvertierung zum Speicherkonzept hinzugepackt hatten, wollten wir auch noch diverse weitergehende Mechanismen anbieten, mit denen der Anwender seine Zwischenablage und die augenblickliche

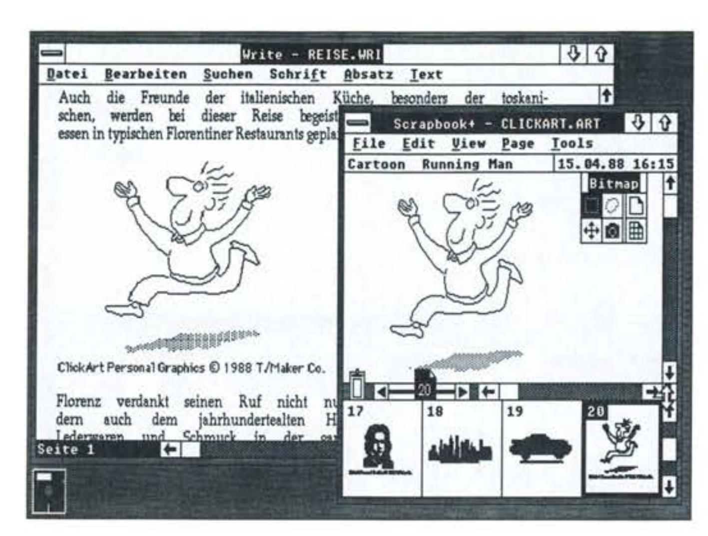


Bild 2: *Ausgewählte Bilder können in andere Windows-Anwendungen eingefügt werden.*

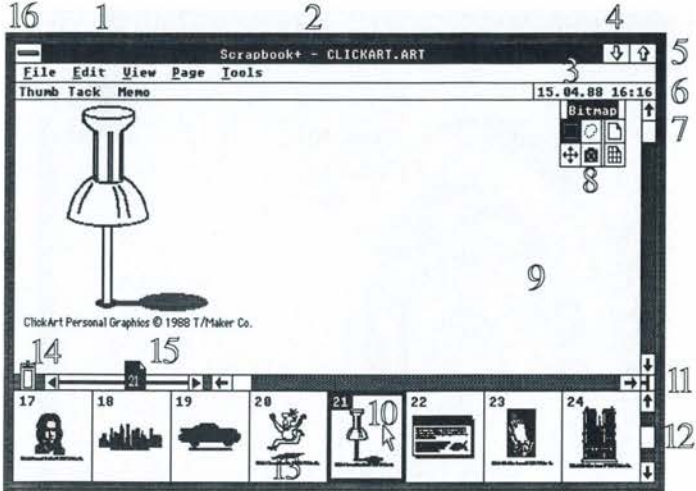


Bild 3: *Die visuellen Bestandteile eines Scrapbook+-Fensters. Mit dem Werkzeugkasten kann man ein Auswahlwerkzeug wählen oder die Art der Anzeige einstellen.*

1 Titelleiste	9 Anzeigebereich
2 Fensterrahmen	10 Horizontale Scrolleiste
3 Menüleiste	11 Größeneinstellung Daumenkino
4 Vergrößern	12 Scrolleiste für Daumenkino
5 Verkleinern	13 Daumenkino
6 Beschreibung	14 Sinnbild Zwischenablage
7 Vertikale Scrolleiste	15 Seitenschieber
8 Werkzeugkasten	16 Steuerungsmenü

Bildschirmanzeige automatisch abspeichern kann. Nach einigen Versuchen mündete das Vorhaben in den Funktionen AutoPaste, ScreenCapture und Camera.

Das Werkzeug zur Datenabfrage

Als gelegentliche Benutzer von PageMaker und Microsoft Excel waren wir von der Unfähigkeit frustriert, an Dinge heranzukommen, ohne die Originalanwendung zu besitzen. Wir wollten in der Lage sein, visuell durch eine große Menge Text, Zeichnungen und Bilder blättern zu können, wenn wir bei der augenblicklichen Arbeit nach einer passenden Ergänzung suchten.

Das Erfordernis, die Daten schnell anzusehen und aussuchen zu können, brachte eine Art Daumenkino hervor, die Funktion Thumbnails als kleine Stellvertreter einer ganzen Seite von Scrapbook+. Das Konzept des Daumenkinos versetzte uns in die Lage, die Informationen zu konzentrieren und große Datenmengen effektiv darzustellen. Mit dem visuellen Index kann man eine spezielle Seite leicht auffinden, die richtige Abbildung leicht auswählen und das resultierende Format in die Zwischenablage kopieren.

Da typische Benutzer sicherlich ihre eigenen Scrapbook+-Dateien erzeugen wollen, gingen wir davon aus, daß sie Zugriff auf die Programme haben, die diese Daten produzieren. So entschieden wir uns also, keine Werkzeuge zum Editieren mit einzubauen. Nebenbei gesagt, einen Editor für alle unterstützten Formate zu entwickeln würde bedeuten, daß viele Funktionen recht dürftig ausfielen. Am Ende haben wir dem Programm dann noch einige eingeschränkte Werkzeuge einverleibt, die ermöglichten, einen Ausschnitt aus den Daten zu wählen: zum Kopieren in die Zwischenablage oder um eine neue Kleindarstellungsform für das Daumenkinos zu erreichen.

Als weitere Verbesserung der Datenabfrage nach der Importierung von Daten direkt von der Platte, hielten wir es für angebracht, ebenso Möglichkeiten zum Export von Daten zu schaffen. Eine logische Folgerung dieser Entscheidung ist, daß die Daten sowohl von der Zwischenablage als auch von der aktiven Scrapbook+-Datei exportiert werden konnten.

Der Vermittler für Daten aus der Zwischenablage

Von Anfang an gab es einen Punkt am Windows, der uns erfreute: die Fähigkeit des einfachen Informationsaustauschs zwischen verschiedenen Anwendungen. Unser Konzept für Scrapbook+ erweiterte diese Anschauung derart, daß der Datenaustausch zwischen Diskette und Anwendung mittels Zwischenablage eingeschlossen wurde, wobei Scrapbook+ als eine Art Vermittler agierte. Beim Basteln an dieser Idee wollten wir erreichen, daß unser Werkzeug einige der lästigen Stolpersteine der Zwischenablage umgehen konnte, während es nahtlos ihre Vorteile übernahm.

Obwohl die momentane Version von Scrapbook+ nicht direkt die entfernte Abfrage und Übertragung von Daten ermöglicht, werden zukünftige Versionen dazu in der Lage sein. Mit der Möglichkeit des dynamischen Datenaustauschs (DDE), könnten Anwendungen eine spezifische Datenstruktur abfragen oder übergeben (submit). Des weiteren könnten Anwendungen Scrapbook+ als Werkzeug zum

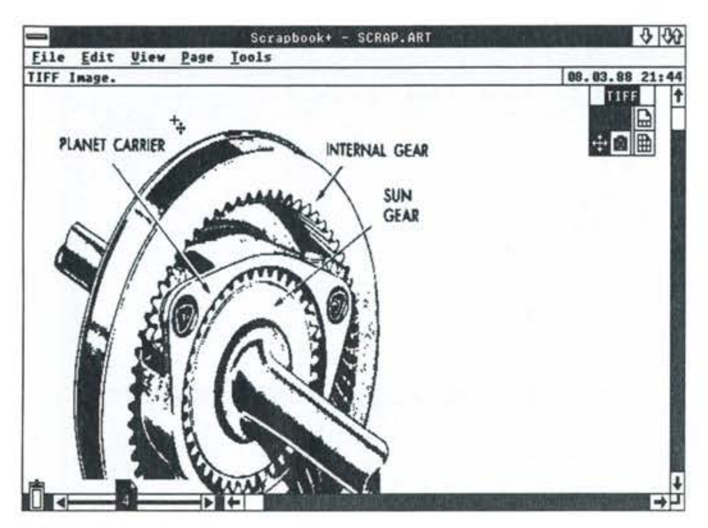


Bild 4: *Nur Anzeige des Bildes.*

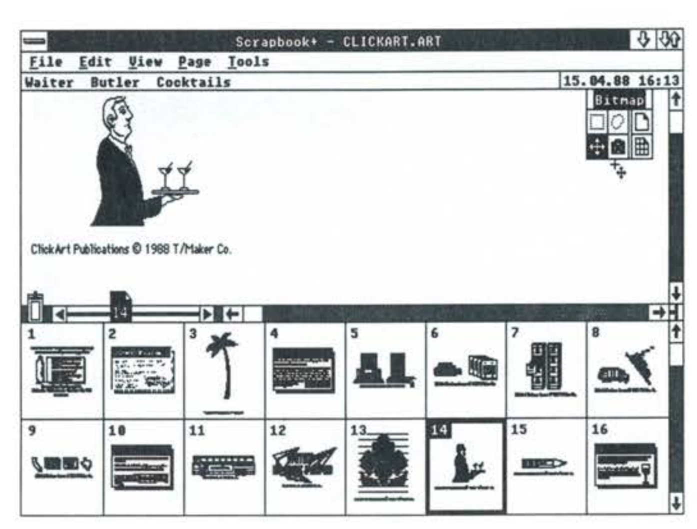


Bild 5: *Anzeige von Bild und Daumenkino.*

Konvertieren von und nach verschiedenen Zwischenablageformaten und externen Diskettendateien benutzen. Auf diese Weise wären viele Entwickler von der lästigen Aufgabe befreit, eine ständig wachsende Zahl von Formaten zu unterstützen, und könnten sich statt dessen auf die Entwicklung neuer innovativer Anwendungen konzentrieren.

Visuelle Komponenten

Scrapbook+ wurde ganz besonders als visuelle Anwendung entworfen. Die meisten Benutzer gehen visuell auf ein Programm ein, und ihr Verständnis wird wesentlich gesteigert, wenn die verwendeten sichtbaren Metaphern die interne Wirkungsweise der Anwendung widerspiegeln.

Beim ersten Start von Scrapbook+ sehen Sie etwa das, was in *Bild 3* dargestellt ist. Obwohl die meisten Komponenten des Scrapbook+-Fensters den erfahrenen Windows-Anwendern vertraut sind, erfordern einige jedoch zusätzliche Erläuterungen. (Die Zahlen weisen auf die im Bild markierten Komponenten hin.)

Der Beschreibungskasten (6) zeigt eine Kurzbeschreibung der aktuellen Seite, inklusive Datum und Zeit, zu dem sie der Scrapbook+-Datei hinzugefügt wurde. Der Werkzeugkasten (8) ist ein bewegliches Unterfenster, das die Auswahl des augenblicklichen Betrachtungsmodus oder eines der vier möglichen Werkzeuge erlaubt. Das Betrachtungsfeld (9) zeigt eine Seite der augenblicklichen Scrapbook+-Datei an und ermöglicht die rechteckige Auswahl eines Teils oder die Wahl mit dem Lasso, um diesen Teil in die Zwischenablage zu kopieren.

Die Größe des Daumenkinos (11) wird mit der Anzahl der im Scrapbook+-Fenster angezeigeten Bilderzeilen bestimmt. Die Scrollleiste (12) ermöglicht das unsichtbare, also im Betrachtungsfeld nicht angezeigte Blättern. Eine Seite im Daumenkino wird durch Anklicken auf das Betrachtungsfeld gebracht. Wenn sie die Anzeige des Daumenkinos (13) wählen, zeigt der untere Teil des Scrapbook+-Fensters ein verkleinertes Abbild jeder Seite der Datei. Obwohl die Abbilder zu klein sein werden, um den Inhalt der Seite im Detail zu sehen, sind sie dennoch groß genug, um die Seiten vergleichen zu können bevor eine zur gezielten Darstellung in das Betrachtungsfeld geholt wird.

Betrachtungsarten

Scrapbook+ arbeitet in zwei Betrachtungsarten, dem Zwischenablage-Modus und dem Datei-Modus. Im Zwischenablage-Modus kann der Inhalt im Betrachtungsfeld angezeigt werden. Sobald die Zwischenablage Informationen enthält, erscheint ein kleines Rechteck in der Mitte des Sinnbilds (14). Wenn die Zwischenablage mehr als ein Format enthält, können Sie durch Herunterziehen des Pull-Down-Menüs View ein anderes für die Anzeige auswählen.

Im Zwischenablage-Modus sind bestimmte Optionen automatisch verfügbar oder nicht zu erreichen. Man kann z.B. den Inhalt der Zwischenablage löschen (um so den Speicher zu entlasten), aber die Auswahlwerkzeuge sind nur erreichbar, während eine Seite der Datei betrachtet wird, nicht aber, wenn man sich die Zwischenablage ansieht.

Zwischen dem Zwischenablage- und dem Datei-Modus kann man durch Anklicken des Seitenschiebers oder durch Auswahl der Scrapbook-Option im Pull-Down-Menü Page umschalten. Im Datei-Modus sind die meisten Menü-Optionen erreichbar, um die Daten zu bearbeiten, abhängig von den vorgegebenen Beschränkungen des gerade betrachteten Formats.

Der Werkzeugkasten wird gebraucht, um ein Auswahlwerkzeug zu wählen oder den aktuellen Betrachtungsmodus zu ändern. Die Auswahlwerkzeuge werden auf der linken Seite und die Modus-Werkzeuge auf der rechten Seite angezeigt. Vier Auswahlwerkzeuge stehen zur Verfügung: der Kasten (block) zur Auswahl rechteckiger Ausschnitte des Betrachtungsfeldes, das Lasso zur Auswahl willkürlicher

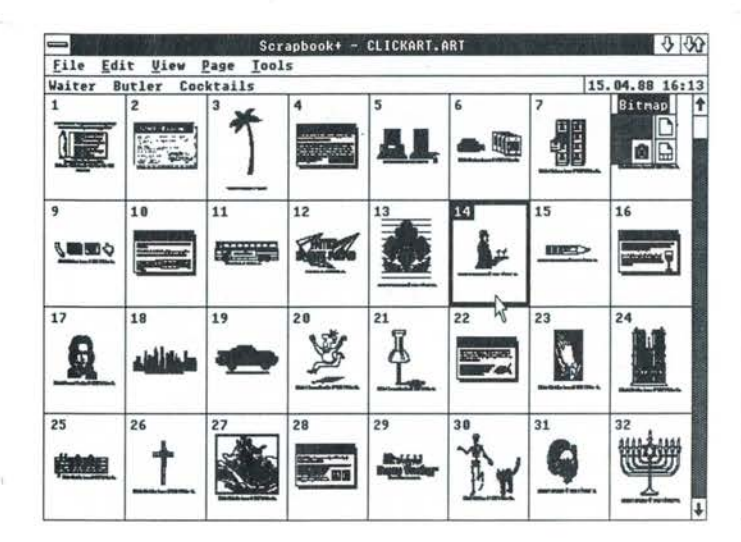


Bild 6: *Nur Anzeige des Daumenkinos.*

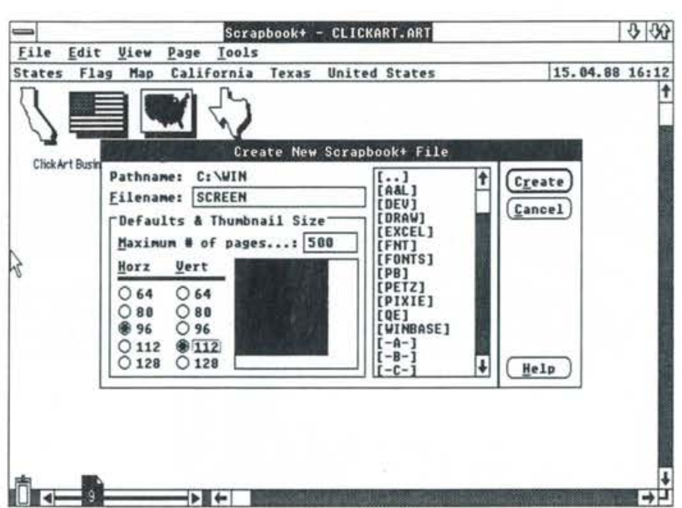


Bild 7: *Im New-Dialogfeld können die Standardwerte eingestellt werden.*

Formen, das schnelle Blättern (scroll) im Betrachtungsfeld und die Kamera zum Festhalten rechteckiger Bildschirmausschnitte in der Zwischenablage.

In jeder der drei Betrachtungsoptionen des Datei-Modus sind nur zwei verschiedene Modus-Werkzeuge verfügbar. Durch Anklicken des gewünschten Modus im Werkzeugkasten verändert sich die Betrachtungsart. *Bild 4* zeigt Scrapbook+ im Modus »Nur Seite«, *Bild 5* zeigt Scrapbook+ im Betrachtungs-Modus »Seite & Daumenkino« und *Bild 6* zeigt den Modus »Nur Daumenkino«.

Der Umgang mit Scrapbook+

Scrapbook+ sollte auf die gleiche Weise zu bedienen sein wie jede andere Windows-Anwendung auch. Wenn Sie das Programm starten, verwendet es automatisch die Datei SCRAP.ART. Eine neue Datei kann jederzeit durch Auswahl der Option New im Pull-Down-Menü File erzeugt werden. Sie können dabei in der sich öffnenden Dialogbox die maximale Anzahl Seiten der erzeugten Datei und die gewünschte Größe des Daumenkinos angeben (*Bild 7*). Es ist wichtig, die Größe jedes Daumenkinos angeben zu können, da Windows eine Menge verschiedener Bildschirmarten unterstützt.

Wurde erst einmal eine Datei erzeugt, kann man die Standard-Bearbeitungsfunktionen benutzen: Ausschneiden (Cut), Kopieren (Copy), Einkleben (Paste) und Löschen (Erase). Im Datei-Modus wirken sie auf die gerade ausgewählte Seite, alle Ablagen eingeschlossen.

Um mehr Flexibilität zu erreichen, haben wir diese standardmäßigen Bearbeitungsfunktionen um spezielle Bearbeitungsbefehle erweitert. Die Optionen des Menüs Special Edit sind normalerweise versteckt und können nur erreicht werden, wenn das Edit-Menü bei gedrückter `Shift`-Taste gewählt wird. Die Standardoptionen Cut, Copy, Paste und Erase ändern sich in Cut Special, Copy Special, Paste Special und Erase Special. Durch Auswahl einer dieser Funktionen wird eine Dialogbox mit den verfügbaren Formaten angezeigt (*Bild 8*), um die für die Operation geeignete einzustellen.

Die Eigenschaft des AutoPaste bezieht sich auf die Standard-Bearbeitungsfunktionen. Wird sie eingeschaltet, dann kommen automatisch alle Veränderungen der Zwischenablage in die aktuelle Scrapbook+-Datei. Als nützlichen Nebeneffekt kann man damit ein Protokoll aller Veränderungen der Zwischenablage erhalten oder eine Reihe von Bearbeitungsbefehlen aufzeichnen, um z.B. große »Blätter« mit Raster-Bildern in einzelne Bilder (Bitmaps) zu zerlegen.

Eine andere Eigenschaft, oft genutzt in Verbindung mit der Funktion AutoPaste, ist die Fuktion ScreenCapture. Mit ScreenCapture kann der Arbeitsbereich, ein Fenster, das ganze Fenster oder auch der ganze Bildschirmbereich in Form einer Bitmap in die Zwischenablage kopiert werden. Mit der Tastenkombination `Alt` `PrtSc` wird der Vorgang eingeleitet, und bei eingeschaltetem AutoPaste kann man leicht Bildschirmabbildungen einfangen, auch Pull-Down-Menüs, sogar wenn Scrapbook+ auf dem Bildschirm zu sehen ist (*Bild 9*).

Eine andere Methode, Bildschirmabbildungen einzufangen, liefert die »Kamera«. Mit ihr läßt sich ein rechteckiger Bereich des Bildschirms auswählen und in die Zwischenablage kopieren, woraus sie dann in die aktuelle Datei gebracht werden kann. Das ist insbesondere für Entwickler, aber auch für Anwender nützlich, da die Kamera erlaubt, alles auf dem Bildschirm sichtbare einzusammeln.

Außer den Standard-Bearbeitungsfunktionen erlauben Ihnen die Import- und Exportfähigkeiten von Scrapbook+, Daten zwischen der aktuellen Datei oder der Zwischenablage und der Diskette zu übertragen. Auf diese Weise kann eine begrenzte Anzahl Konvertierungen zwischen den Standardformaten der Zwischenablage und einigen gängigen Dateiformaten durchgeführt werden (*Tabelle 1*).

Format	Block	Lasso	Scroll	Import	Export
Text			■	txt	txt
Rich Text			■	rtf	rtf
Bitmap	■	■	■	msp,bmp	msp,bmp
Drucker-Bitmap	■	■	■		msp,bmp
Grafik		■	wmf		wmf
Druckergrafik			■		wmf
CSV			■	csv	csv
SYLK			■	syk	syk
DIF			■	dif	dif
TIFF			■	tif	tif
PostScript			■	eps,ps	eps,ps

Tabelle 1A: *Nicht für jedes Format der Zwischenablage sind alle Werkzeuge verfügbar.*

txt	Nahezu alle Textprogramme
rtf	Viele Texverarbeitungsprogramme
msp	Windows Paint
bmp	Windows Bitmap-Editor
wmf	Einige Zeichenprogramme
csv	Die meisten Tabellenkalkulationen und Datenbanken
slk	Microsoft Excel, Multiplan
dif	Die meisten Tabellenkalkulationen und Datenbanken
tif	Scanner, andere Anwendungen
eps	Adobe Illustrator
ps	Adobe Illustrator

Tabelle 1B: *Die Dateitypen einiger üblicher Quellen, die von Scrapbook+ unterstützt werden.*

Eine für Windows-Entwickler interessante Sache ist die Unterstützung des BMP-Dateiformats. Mit Scrapbook+ kann man große Bitmap-Bibliotheken erhalten (größere als die vom Icon-Editor unterstützten) und man kann ausgewählte zur Einbindung in die Recourcen-Datei einer Anwendung exportieren.

Eine der sehr nützlichen Funktionen von Scrapbook+ heißt Merge (*Bild 10*) mit der man Teile oder komplette Scrapbook+-Dateien in anderen Dateien kombinieren kann. Das ist besonders hilfreich, wenn große Dateien aus vielen kleinen zusammengesetzt werden sollen.

Herausforderungen bei der Entwicklung

Ein Programm wie Scrapbook+ in einer neuen und herausfordernden Umgebung wie Windows zu entwickeln, bedeutete, auf viele interessante und schwierige Probleme zu stoßen. Eine der größten Herausforderungen bei der Entwicklung war, schnellen Zugriff auf eine große Anzahl Seiten zu ermöglichen, ohne unverschämt viel Speicher zu

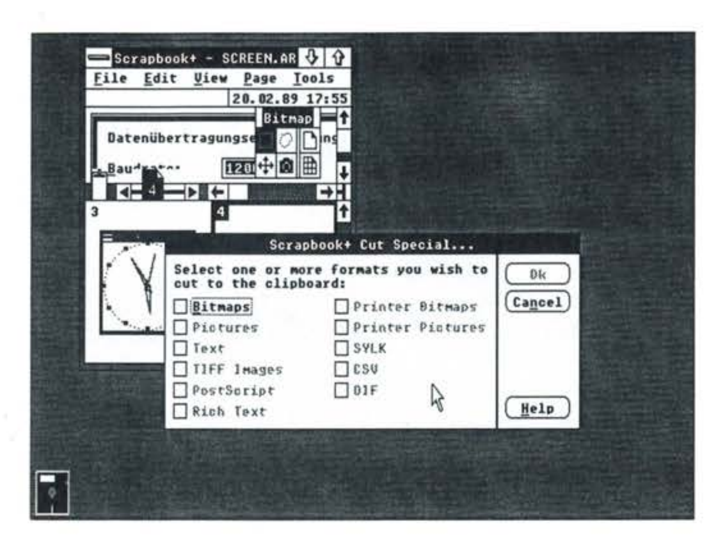


Bild 8: *Im Dialogfeld Special Edits wird das Format für das Auschneiden und Kopieren in die Zwischenablage eingestellt.*

nutzen. Jede Scrapbook+-Datei besteht aus einem Header mit fester Länge, gefolgt von einem Index mit variabler Größe mit einem Eintrag je Seite, gefolgt von den Daten der Seiten selbst. Um schnellen Zugriff zu erreichen, halten wir den Header und den Index im Speicher. Eine 500-Seiten-Datei benötigt weniger als 8 Kbyte Speicher.

Die Fähigkeit, eine Datei auf schnelle Weise »grafisch durchzulesen«, war eine andere wichtige Forderung beim Entwurf, die zu dem Konzept des Daumenkinos führte. Zuerst entschieden wir, daß das Daumenkino beim Einfügen einer Datei erzeugt und auf Diskette geschrieben werden sollte. Wir wollten das Daumenkino schnell anzeigen und dennoch Hauptspeicher erhalten. Unsere Lösung des Problems war die Benutzung entfernbarer Bitmaps für das Daumenkino. Windows kann sie dann bei Speicheranforderungen entfernen, und es ist trotzdem ein schnelles Blättern im Daumenkino möglich, besonders beim Zurückblättern auf Abbildungen, die ins Daumenkino eingelesen und noch nicht entfernt wurden.

System-Abhängigkeiten

Ein großer Vorteil der Windows-Umgebung ist ihre Unabhängigkeit von den benutzten Geräten. Obwohl die für Windows entwickelte Software auf vielen Bildschirmen und unter unterschiedlichsten Speicherkonfigurationen läuft, inklusive auf Rechnern mit Expanded Memory, ist der Programmierer nicht vollkommen unabhängig.

Scrapbook+ mußte getestet und auf die gebräuchlichsten Monitore und Computer-Konfigurationen eingestellt werden, um sicherzustellen, daß jede Ressource die passenden Eigenschaften hatte und die Speicherverwaltung korrekt arbeitete. Auch mußte eine umfassende Fehlerbehandlung eingebaut werden, um Situationen in den Griff zu bekommen, in denen der Videoadapter nicht kompatibel zu den von Scrapbook+ verwalteten Daten war.

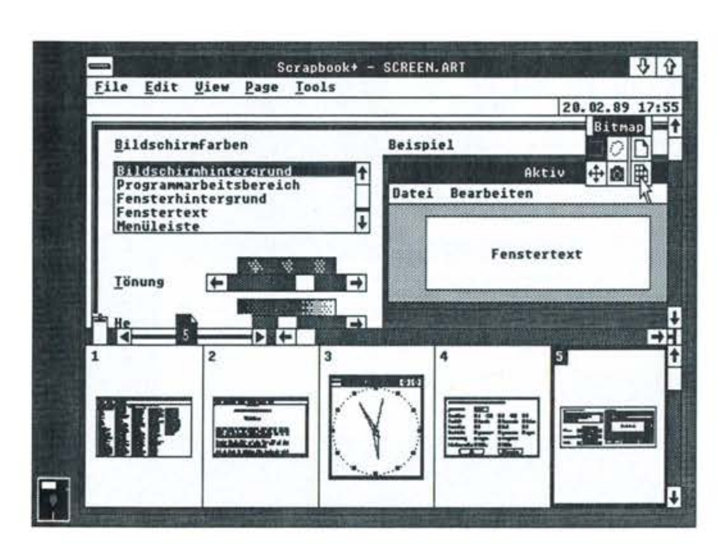


Bild 9: *Mit Scrapbook+ können der ganze Bildschirm, das aktive Fenster oder der Anzeigebereich des aktuellen Fensters gespeichert werden.*

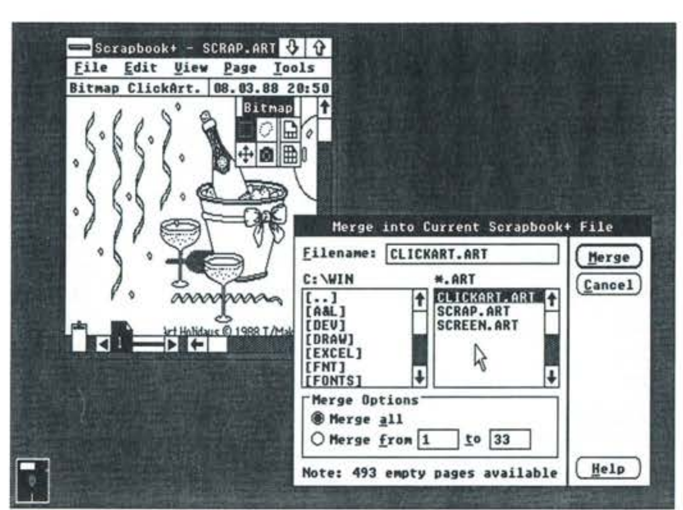


Bild 10: *Mit dem Befehl Merge können Teile einer Scrapbook+-Datei oder auch die ganze Datei in einer anderen Datei gespeichert werden.*

Die Entwicklung von Scrapbook+ begann unter der Windows-Version 1.03; später haben wir die Version 2.03 benutzt. Der Wechsel von den nebeneinander liegenden Fenstern zu den überlappenden erforderte einige Änderungen unseres Entwurfs. Auch bestärkte uns die formale Definition der Tastatur-Hooks und die Unterstützung des Expanded Memory, bedeutende Änderungen an der Funktion Screen Capture vorzunehmen. Das schloß die Entwicklung einer dynamischen Link-Bibliothek für unseren Tastatur-Hook der Screen Capture-Funktion ein, um die Auslagerung unter dem Expanded Memory zu verhindern.

Vielfältigkeit der Formate

Unser ursprüngliches Entwicklungsziel bei Scrapbook+ war es, jeden Datentyp der Zwischenablage behandeln zu können. In der Praxis erwies sich dies als sehr schwierig und in einigen Fällen sogar als unmöglich. Einige Formate der Zwischenablage wie die Besitzeranzeige und verschiedene Eigentümer-Formate hatten nun mal keine Bedeutung ohne den Zusammenhang mit dem Programm, das sie in die Zwischenablage kopierte. Andere Formate wie Text, Bitmaps und Metadateien bildeten die Grundlage für den Datenaustausch mit der Windows-Zwischenablage, und es bestand nie ein Zweifel über ihre Nützlichkeit.

In der frühen Entwicklungsphase von Scrapbook+ verpflichteten wir uns selbst, Encapsulated PostScript (EPS) zu unterstützen. Da wir die ersten waren, die mit diesem Format umgingen, mußten wir ein Standardprotokoll für den Austausch von EPS-Bildern über die Zwischenablage definieren. Für diejenigen, die nicht mit diesem Format vertraut sind: EPS besteht aus einer Kombination von PostScript-Text mit einer optionalen Anzeige-Beschreibung des Bildes, die eine Metadatei, eine TIFF-Datei (Tagged Image File Format) oder PICT-Datei vom Macintsoh sein kann.

Unsere Verpflichtung gegenüber EPS führte uns zur Unterstützung von TIFF, etwas, was wir im Innersten hofften, vermeiden zu können, da es das Projekt bedeutend verzögern konnte. Jedoch wurde, einmal abgesehen von der Komplexität dieses Formats, die Unterstützung von EPS und TIFF zum wichtigsten Verkaufsargument des Programms - das hatten wir nicht vorausgesehen.

Zusammenspiel mit Anwendungen

Eine Phase der Entwicklung, die mehr Zeit in Anspruch nahm als wir gedacht hatten, war das Zusammenspiel von Scrapbook+ und anderen Windows-Programmen. Viele Spitzfindigkeiten und Verflechtungen von Windows kamen in diesem Bereich zu Tage. Wir lernten viel über die Korrekte Benutzung der Zwischenablage, indem wir sicherstellten, daß Scrapbook+ mit vielen anderen Windows-Programmen richtig zusammenarbeitete, inklusive Microsoft Excel, Aldus PageMaker und Micrografx Designer.

Wir waren der Ansicht, die Anwender würden immer stärker danach verlangen, daß Programme nicht nur für sich allein gut laufen, sondern auch im Zusammenspiel mit anderen Anwendungen. Deshalb haben wir viel Zeit darauf verwandt, das Zusammenspiel zwischen Scrapbook+ und den anderen wichtigen Windows-Programmen zu studieren, wobei wir besonderes Augenmerk auf die Zusammenarbeit mit der Zwischenablage und den Speicherverbrauch hatten.

Am Ende, und nicht unerwartet, arbeitete Scrapbook+ beträchtlich besser, als wir alle Codesegmente, auch _TEXT, kleiner als 8 Kbyte hielten und die Datensegmente auf ein Minimum reduzierten.

Optimierung der Performance

Da wir Scrapbook+ als eine System-Utility sahen, die fast zu jeder Zeit auf dem Rechner gegenwärtig sein würde, unternahmen wir einige Anstrengungen, um die Leistung des Programms zu bestimmen und in bestimmten kritschen Punkten zu verbessern.

Wir haben sowohl statische als auch dynamische Modelle mit verschiedenen selbstentwickelten Code-Analysetechniken benutzt, um der Anwendung ein gutes mathematisches Modell zugrundezulegen. Dies Modell wurde optimiert und alle Kraft auf die kritischen Bereiche konzentriert. Das Ergebnis war einen beträchtlich verbesserte Anwendung mit ausgeglichenen Segmenten. In bestimmten Situation wurde die für einige Operationen benötigte Zeit um den Faktor fünf reduziert.

Entwicklungswerkzeug

Während der Arbeit an Scrapbook+ haben wir das Programm extensiv als Enwicklungswerkzeug für die Behandlung von Bitmaps, Cursorn und Sinnbildern herangezogen. Jede Verkörperung dieser Ressourcen wurde zum Betrachten und zum möglichen Einsatz in einer einzelnen Datei abgespeichert. Mit dem Daumenkino konnten wir die letzten Entwürfe schnell mit älteren, aus einer früheren Entwicklungsphase, vergleichen.

Zusätzlich zum Nutzen als Entwicklungswerkzeug benutzten wir Scrapbook+ in vielen Fällen, um komplizierte Wechselwirkungen der Zwischenablage mit sich selbst oder mit anderen Anwendungen zu testen. Darüber hinaus war es bei der Dokumentation des Programms ein Hilfsmittel unschätzbaren Werts. Mit ihm haben wir eine etwa 1,5 Mbyte große Datei erzeugt, die jede Illustration, jedes Bild, jede Bildschirmabbildung des Handbuchs und jedes Sinnbild enthielt.

Im Rückblick

Da nun die erste Version von Scrapbook+ fertiggestellt ist, sind einige Betrachtungen über den ganzen Prozeß angebracht. Zuallererst sind wir uns der großen Herausforderung, die die Programmierung von Windows-Software in sich birgt, noch bewußter geworden. Obwohl teilweise auf-

grund der Komplexität der Umgebung, haben die Anforderungen an Software einen Punkt erreicht, den wir nur noch mit großen Mühen erreichen können.

Wir sind weiterhin fest davon überzeugt, daß die Programmierung in großen Modulen ein Überbleibsel vergangener Zeiten ist, und daß die Software der Zukunft aus kleinen vollständigen Paketen besteht, die von den Benutzern in eine funktionelle Umgebung eingebunden werden.

Wir haben auch gelernt, wie sehr wir geneigt waren, die Phase der Qualitätssicherung bei der Entwicklung einer Windows-Anwendung zu unterschätzen. Die Eigenschaften der gemeinsam genutzten Ressourcen und das Multiprocessing von Windows machen es schwer, Anwendungen zu testen, und einfache Fehler werden zu komplizierten Wechselwirkungen, deren Lösung Wochen dauern kann.

Schließlich lernten wir, wie hinterlistig die Fallen von »eingeschlichenen Eigenschaften« sein können. Man sagt so leicht: »Laßt uns noch dies oder jenes mit einbauen, es dauert nur unwesentlich länger.« Das Problem ist jedoch, das jedes dieser Features seinen Tribut fordert und sich das Auslieferungsdatum des Produkts immer weiter verschiebt.

Wenn man alle Dinge betrachtet, dann hat sich Scrapbook+ zu einem Programm gemausert, das wir in der Art nicht erwartet hatten. Bei vielen Gelegenheiten haben wir uns gefragt: »Wie haben wir das bloß in den Griff bekommen?« Würden wir so etwas noch einmal machen? Ja, sofort, sobald wir wieder eine großartige Idee haben.

Kevin P. Welch, David E. West

MS-Word Dateien lesen, erstellen und manipulieren

Das Dateiformat von Microsoft Word 4.0

Obwohl MS-Word ganz ohne Zweifel zu den populärsten PC-Programmen überhaupt zählt, bieten bisher nur ganz wenige Softwarehäuser sogenannte Add-On-Produkte für dieses erfolgreiche Textverarbeitungsprogramm an, während der Markt gerade in den USA mit solchen Produkten für andere Programme, wie etwa dBase oder Lotus 1-2-3, geradezu überschwemmt wird. Das Fehlen solcher Produkte für MS-Word liegt sicher nicht zuletzt an dem bisher undokumentierten Dateiformat, das aufgrund der Leistungsfähigkeit dieses Programms eine Komplexität aufweist, die es einem Programmierer ohne weitere Hinweise fast unmöglich macht, sie zu entschlüsseln.

Erstmalig veröffentlichen wir deshalb in dieser Folge des *Microsoft System Journals* das Dateiformat von MS-Word und geben Ihnen damit die Möglichkeit, MS-Word-Dateien mit eigenen Programmen zu verarbeiten oder Dateien für die Bearbeitung durch MS-Word zu erstellen. Dabei soll nicht nur die Theorie, also der Aufbau von MS-Word-Dateien im Vordergrund stehen. Vielmehr stellen wir ein nützliches Programm vor, mit dessen Hilfe Hardcopy-Dateien in Word-Dateien umgewandelt und Hardcopies dadurch direkt in den Text integriert werden können.

Bei unseren Betrachtungen beziehen wir uns auf das Format der MS-Word-Version 4.0. Zwar weist die in den USA bereits angekündigte Version 5.0 demgegenüber einige, bisher undokumentierte Erweiterungen auf, doch spielen diese Veränderungen keine Rolle, solange Sie MS-Word-Dateien nur lesen oder erstellen. Eine Modifikation von Word-5.0-Dateien sollte jedoch unterbleiben, da in diesem Fall Probleme auftreten können, die in der Regel zum Absturz von MS-Word führen.

Aufbau des Vorspanns in der Page Ø			
Ofs.	Name	Bedeutung	Typ
Ø	wIdent	Identitätscode, muß dezimal 48689 enthalten	1 WORD
2	dty	reserviert, muß Ø enthalten	1 WORD
4	wTool	Identitätscode, muß dezimal 43776 enthalten	1 WORD
6	wDummy	reserviert, müssen Ø enthalten	4 WORD
14	fcMac	Anzahl der Zeichen im Text plus 128	1 DWORD
18	pnPara	Page-Nummer mit der die Paragraphen-Formate beginnen	1 WORD
2Ø	pnFntb	Page-Nummer mit der die Fußnotentabelle beginnt	1 WORD
22	pnSep	Page-Nummer mit der die Bereichsformate beginnen	1 WORD
24	pnSctb	Page-Nummer mit der die Bereichs-Tabelle beginnt	1 WORD
26	pnPgtb	Page-Nummer mit der die Seitenumbruchs-Tabelle beginnt	1 WORD
28	pnSumd	Page-Nummer der Inhaltsangabe (Inhalt, Autor etc.)	1 WORD
3Ø -95	szSsht	kompletter Dateiname der Druckformatvorlage als C-String	66 BYTE
96	pnMacWrite	reserviert für Windows Write	1 WORD
98 -1Ø5	szPrd	Name des Druckertreibers ohne Erweiterung und Pfad als C-String	8 BYTE
1Ø6	pnMac	Anzahl der Pages in der Datei	1 WORD
1Ø8	bfRev	Flag für die Markierung überarbeiteter Textteile (s.u.)	1 WORD
11Ø -127	wDummy1	reserviert, müssen Ø enthalten	28 BYTE

Abbildung 1: *Der Vorspann einer Word-Datei in der Page 0.*

Der Aufbau einer Word-Datei

Eine Word-Datei unterteilt sich in drei Bereiche:

- den Vorspann,
- den eigentlichen Text und
- die Format-Informationen (Zeichen-, Seiten-, Bereichs- und Absatz-Formate, Fußnotenverweise und die Informationen über Textinhalt, Autor etc.)

MS-Word unterteilt die Datei dabei in einzelne »Pages« (Seiten), die jeweils 128 Bytes umfassen. Jeder Bereich beginnt mit dem Anfang einer Page und nimmt mindestens eine Page in Anspruch, auch wenn er sie nicht ganz ausfüllt. Dadurch werden zwar unter Umständen einige Bytes verschenkt, doch ist es im Hinblick auf die Performance grundsätzlich effizienter, mit festen Satzlängen zu arbeiten, als die einzelnen Bereiche unmittelbar aneinanderzuhängen.

Der Vorspann

Die erste Page innerhalb einer MS-Word-Datei, also die Bytes mit den Offsetadressen 0 bis 127, nimmt den Vorspann auf. Er enthält einige spezielle Codes, mit denen sich die Datei als eine MS-Word-Datei zu erkennen gibt, sowie Zeiger auf die Adressen bzw. Page-Nummern der einzelnen Bereiche innerhalb der Datei. Die *Abbildung 1* zeigt den Aufbau des Datei-Vorspanns.

Vielleicht kommen Ihnen die Namen der einzelnen Felder innerhalb des Vorspanns »spanisch« vor, da sie sich so ganz von der gewohnten Namensgebung für Variablen unterscheiden. Es sind jedoch die Original-Namen, die die Entwickler von MS-Word diesen Feldern gegeben haben, wobei sie sich weniger an das »Spanische«, als an das »Ungarische« gehalten haben. So nämlich werden die

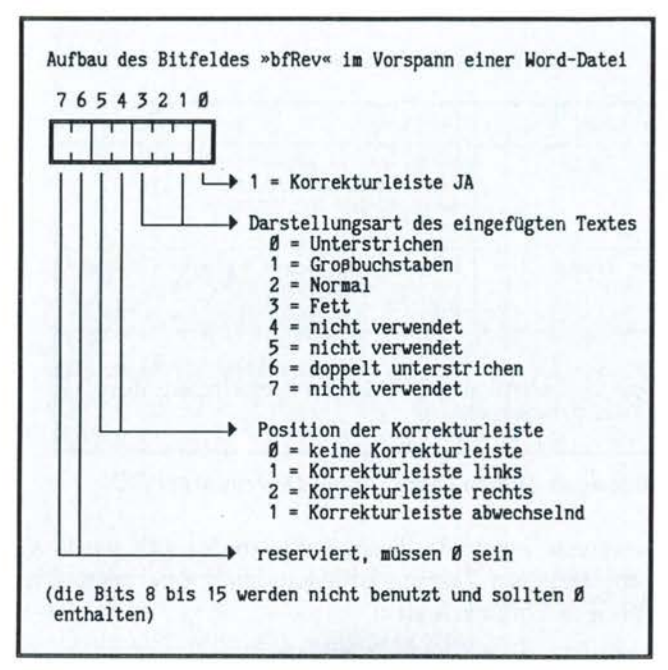


Abbildung 2: *Aufbau des Feldes* `bfRev` *im Vorspann einer Word-Datei.*

Regeln zur Bildung von Variablennamen bezeichnet, die innerhalb der »Applications Development Group« von Microsoft eingesetzt werden.

Ihren Namen verdanken diese Regeln zum einen der Nationaltität ihres Urhebers, Charles Simonyi, und zum anderen ihrer Undurchsichtigkeit für Außenstehende, die dadurch leicht auf den Gedanken kommen können, die Namen seien einer unbekannten Fremdsprache, wie etwa Ungarisch entliehen. Leider fehlt uns an dieser Stelle der Platz, um detaillierter auf die Regeln einzugehen, die der Bildung dieser Variablennamen zu Grunde liegen, doch wird sich eventuell ein Artikel in einer der nächsten Folgen des Microsoft System Journals mit diesem Thema beschäftigen.

Kommen wir jedoch zurück zum Vorspann unserer MS-Word-Datei. Beachten Sie bitte, daß die Reihenfolge der verschiedenen Felder innerhalb des Vorspanns, die den Anfang eines der verschiedenen Bereiche innerhalb der Datei angeben (`pnPara`, `pnFtb` etc.), auch die Reihenfolge dieser Bereiche innerhalb der Datei widerspiegelt. Die Reihenfolge lautet also: Vorspann, Text, Absatzformate, Fußnotentabelle, Bereichsformate, Bereichstabelle, Seitenumbruchs-Tabelle und schließlich Inhaltsangabe.

Wie wir jedoch später noch sehen werden, muß eine MS-Word-Datei nicht unbedingt alle diese Komponenten enthalten, da es zum Beispiel keinen Sinn macht, eine Fußnotentabelle anzulegen, wenn im Text gar keine Fußnoten definiert werden.

Wie auch viele andere der im folgenden vorgestellten Strukturen, enthält der Vorspann ein Feld, dessen Inhalt nicht als Ganzes, sondern Bit für Bit interpretiert werden muß. Solche Variablen bezeichnet man als Bitfelder, weshalb man in ihren Namen immer die Buchstabenkombination `bf` findet. Das Flag `bfRev` (Offset 108) ist ein solches Bitfeld, das angibt, ob und wie überarbeitete Textteile gekennzeichnet werden sollen (*Abbildung 2*).

Der Text

Innerhalb der Word-Datei beginnt der eigentliche Text immer in der Page 1, also ab der Offsetadresse 128 innerhalb der Datei. Die Anzahl der gespeicherten Zeichen, wie auch die Anzahl der von ihnen belegten Pages ergibt sich aus dem Feld `fcMac` innerhalb des Vorspanns. Die Formel zur Berechnung der Anzahl der belegten Pages lautet:

```
( fcMac - 1 ) / 128
```

Innerhalb dieser Pages wird der Text fortlaufend gespeichert, wobei je nach der Länge des Textes ein mehr oder minder großer Teil der letzten Page ungenutzt bleibt. Die einzelnen Zeichen werden im erweiterten IBM-ASCII-Zeichensatz gespeichert, wobei der Text keinerlei Formatinformationen enthält. Lediglich den folgenden Zeichen- bzw. Zeichenkombination ordnet MS-Word eine besondere Bedeutung zu:

13,10 **Absatzende**. Die beiden Zeichen dürfen nur in Verbindung miteinander auftauchen und markieren grundsätzlich das Ende eines Absatzes.

11 **Zeilenumbruch**, der durch Betätigung von [Shift][Return] eingegeben wurde.

12 **Seitenumbruch**, der durch Betätigung von [Shift] [Ctrl] [Return] eingegeben wurde und Seitenvorschub, den MS-Word bei der Formatierung erstellt hat.

9 **Tabulator**

32 **ungeschütztes Leerzeichen**

255 **geschützter Wortzwischenraum**, Eingabe durch [Ctrl][Space].

45 **wahlweiser Trennstrich**

196 **geschützter Trennstrich**, Eingabe durch [Ctrl][Shift][-] (im IBM-ASCII-Zeichensatz entspricht dieses Zeichen ebenfalls dem '-'-Zeichen. Die Interpretation dieses Zeichens durch MS-Word ist deshalb kontextsensitiv.)

31 **bedingte Trennstriche**, Trennstriche, die von MS-Word innerhalb des Bibliotheksdienstes »Silbentrennung« angelegt oder mit [Ctrl][-] eingegeben werden.

Eine besondere Bedeutung kommt auch den folgenden Codes zu, wenn das Zeichen innerhalb der Zeichen-Formatbeschreiber als ein Zeichen mit dem Format `fSpecial` gekennzeichnet wird. Die einzelnen Codes entsprechen dann den verschiedenen Textbausteinen, die MS-Word grundsätzlich zur Verfügung stellt:

1 *(Seite)*
2 *(Druckdatum)*
3 *(Druckzeit)*
4 *(Fußnote)*

Aufbau einer Page mit Zeichenformaten			
Ofs.	Name	Bedeutung	Typ
Ø	fcFirst	Nummer des ersten Zeichens, dessen Format innerhalb dieser Page beschrieben wird*	1 DWORD
4 - i	rgfod	ein Array mit FODs (Format-Descriptors)	x * FOD
i+1 -126	grpfprop	eine Aneinaderreihung von FPROPs (Format Properties)	variabel
127	cfod	Anzahl der FODs in dieser Page	1 BYTE

*das erste Zeichen innerhalb des Textes hat die Nummer 128

Abbildung 3: *Aufbau einer Page mit Zeichenformaten.*

Zeichenformate

Der Bereich der Zeichenformate beginnt in der Page unmittelbar hinter dem Text. Die zugehörige Page-Nummer läßt sich dadurch aus der Anzahl der Zeichen errechnen, die innerhalb des Vorspanns gespeichert sind. Die Formel dafür lautet:

```
pnChar = ( fcMac + 127 ) / 128
```

Die Zeichenformate beschreiben nicht das Format jedes einzelnen Zeichens, sondern das Format jeweils einer Gruppe von Zeichen, die über ein identisches Zeichenformat verfügen. Identisch bedeutet in diesem Zusammenhang, daß die Zeichenformate bis in das kleinste Detail übereinstimmen. Wenn ein Text aus Zeichen mit einem identischen Format besteht, aber ein Zeichen in der Mitte des Textes zusätzlich das Attribut »Fett« trägt, müssen bereits drei Formatbeschreiber, sogenannte *Format Descriptors*, angelegt werden. Einer für die Zeichen, bis zu dem Zeichen mit dem Attribut »Fett«. Einer für dieses Zeichen und schließlich einer für die Zeichen bis zum Ende des Textes. Wie die *Abbildung 3* zeigt, kann eine Page dabei mehrere Formatbeschreiber aufnehmen.

Wie hier ebenfalls zu sehen ist, befinden sich nur zwei Informationen innerhalb einer Page mit Zeichenformaten an einer fest lokalisierbaren Adresse: die Felder `fcFirst` und `cfod`. Ersteres gibt die Nummer des ersten Zeichens innerhalb des Textes an, das durch diese Page beschrieben wird. Daß die Zeichenpositionen innerhalb des Texts nicht ab 0, sondern ab 128 gezählt werden, hängt mit dem Aufbau der Datei zusammen. Wie wir eben gesehen haben, beginnt der eigentliche Text mit der zweiten Page innerhalb der Datei und damit ebenfalls mit der Offsetadresse 128. Durch das Fortschreiben der Zeichenpositionen ab 128 wird nun erreicht, daß die angegebene Zeichenposition jeweils auch der Offsetposition des Zeichens innerhalb der Datei entspricht. Dadurch läßt sich sowohl die Page, als auch die

Aufbau eines »Format Descriptors« (FOD)			
Ofs.	Name	Bedeutung	Typ
Ø	fcLim	Nummer des Zeichens hinter dem letzten Zeichen, das durch diesen FOD beschrieben wird*	1 DWORD
2	bfprop	Entfernung zwischen »rgfod« und dem zugehörigen FPROP in Bytes**	1 WORD

*das erste Zeichen innerhalb des Textes trägt die Nummer 128
** der Wert ØFFFFh zeigt an, daß das Zeichenformat nicht vom Standardformat abweicht

Abbildung 4: *Aufbau eines Format Descriptors (FOD).*

Offsetadresse innerhalb dieser Page, an der sich das bzw. die beschriebenen Zeichen befinden, leicht errechnen. Die zugehörigen Formeln lauten:

```
PageNummer = Position / 128
PageOffset = Position modulo 128
```

Da die einzelnen Formatbeschreiber eine variable Länge besitzen, paßt nicht immer die gleiche Anzahl von ihnen in eine Page. Deshalb wird ihre Anzahl jeweils innerhalb der Variablen `cfod` angegeben. Damit wird auch die Anzahl der Formatbeschreiber (`FODs`) im Array `rgfod` festgelegt. Die einzelnen `FODs` haben dabei folgenden Aufbau.

Die *Abbildung 4* macht deutlich, daß der Format Descriptor seinem Namen keine Ehre macht, da er das Format der Zeichen nicht selbst beschreibt, sondern lediglich einen Verweis auf einen sogenannten `FPROP`, auf einen *Formatting Property* enthält, der das Format der Zeichen festlegt. Dies macht durchaus Sinn, denn dadurch kann innerhalb der Page eine Menge Speicherplatz gespart werden, wenn die Zeichen verschiedener Bereiche ein identisches Format aufweisen.

Dies ist zum Beispiel in dem oben angeführten Beispiel der Fall, da sowohl die Zeichen vor dem Zeichen mit dem Attribut »Fett«, als auch die Zeichen dahinter über ein identisches Format verfügen. Sofern die zugehörigen Format Properties in eine Page passen (und davon kann man ausgehen), müssen innerhalb dieser Page lediglich zwei von ihnen angelegt werden. Einer für die Zeichen vor und hinter dem »fetten« Zeichen und einer für dieses Zeichen selbst.

Innerhalb von `rgfod` verweisen dann der erste und der dritte Eintrag auf ein und denselben Format Property. Beachten Sie aber bitte, daß dies nur in solchen Fällen möglich ist, in denen sich mehrere Zeichenbereiche mit identischem Format innerhalb einer Page befinden, da das Feld `bfprop` innerhalb eines `FODs` nicht aus einer Page hinausweisen darf.

Eine einwandfreie Zuordnung der verschiedenen `FPROPs` zu den einzelnen Zeichenbereichen gewährleistet das Feld `fcLim` innerhalb des `FOD`. Es gibt die Nummer

Aufbau eines »Formatting Properties« (FPROP) innerhalb der Zeichenformate

Ofs.	Name	Bedeutung	Typ
Ø	cch	Anzahl der Bytes in diesem FPROP exklusive dieses Bytes	1 BYTE
1	chpbf1	Code des Zeichenformats aus der Druckformatvorlage*	1 BYTE
2	chpbf2	Zeichenformat und Zeichensatznummer*	1 BYTE
3	hps	Größe des gewählten Zeichensatzes in halben Punkten	1 BYTE
4	chpbf3	weitere Zeichenattribute*	1 BYTE
5	hpsPitch	reserviert, muß Ø enthalten	1 BYTE
6	hpsPos	horizontale Zeichenposition**	1 BYTE
7 -1Ø	fpres	reserviert, müssen Ø enthalten	4 BYTE

*hier handelt es sich um Bitfelder, die auf den folgenden Seiten beschrieben werden
**Ø steht für normal, 1-127 für hochgestellt, 128-255 für tiefgestellt

Abbildung 5: *Aufbau eines Formatting Properties (FPROP) innerhalb der Zeichenformate.*

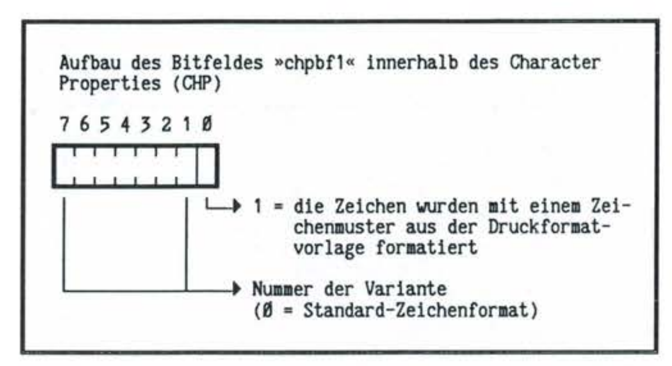


Abbildung 6: *Aufbau des Bitfeldes* chpbf1 *innerhalb des Character Properties (CHP).*

des Zeichens hinter dem letzten Zeichen an, auf das sich der jeweilige FOD bezieht. Für den ersten FOD sind dies die Zeichen zwischen fcFirst *(Abbildung 3)* und fcLim-1 im ersten FOD. Der zweite FOD beschreibt dann die Zeichen zwischen fcLim aus dem ersten FOD und seinem eigenen fcLim-1. Allgemein läßt sich daher festhalten:

$\mathtt{StartZeichen}_n$ = fcFirst für n=Ø und $\mathtt{fcLim}_{(n-1)}$ für n<>Ø

$\mathtt{EndZeichen}_n = \mathtt{fcLim}_{n-1}$

Da die einzelnen FODs die verschiedenen Zeichenbereiche grundsätzlich in aufsteigender Reihenfolge beschreiben, die erste dieser Pages mit dem ersten Zeichen innerhalb des Textes beginnt und alle darauffolgenden Pages jeweils an das Ende der vorangegangenen Page anknüpfen, markiert das Feld fcLim im letzten FOD einer Page gleichzeitig das erste Zeichen in der nächsten Page und damit den Inhalt des Feldes fcFirst in der darauffolgenden Page.

Beachten Sie bei der Erstellung oder Manipulation von MS-Word-Dateien bitte, daß jedes Zeichen innerhalb des Textes durch einen FOD beschrieben werden *muß* und das Feld fcLim im letzten FOD innerhalb der letzten Page der Zeichenformate auf das Zeichen hinter dem letzten Zeichen im Text zeigen muß. Bereits ein kleiner Fehler kann dabei den Absturz von MS-Word zur Folge haben, wenn es die Zeichenformate nicht in der erwarteten Anordnung vorfindet.

Kommen wir zurück zu den Zeichenformaten und damit zu den FPROPs, die diese Formate beschreiben. Während das Array rgfod mit jedem Eintrag auf das Ende der Page zuwächst, wachsen die FRPROPs vom Ende der Page in Richtung auf das Ende von rgfod. Das Ende des ersten FPROPs, also des Formatbeschreibers, der durch rgfod[Ø] beschrieben wird, grenzt dadurch unmittelbar an das Feld cfod am Ende der Page. Der zweite FPROP (rgfod[1]) endet unmittelbar vor dem ersten FPROP. In dieser Weise »folgen« die weiteren FPROPs, bis nicht mehr genug Platz zwischen dem Ende des Arrays rgfod und dem letzten FPROP ist, um einen weiteren FPROP in die Page aufzunehmen. Den Aufbau eines solchen FPROPs zeigt die *Abbildung 5.*

Eine besondere Bedeutung innerhalb eines FPROPs kommt dem Feld cch zu. Um Platz zu sparen und dadurch möglichst viele Zeichenbereiche innerhalb einer Page beschreiben zu können, werden nämlich nur so viele Bytes des FRPOPs gespeichert, wie »unbedingt notwendig« sind. Mit diesem Terminus werden dabei alle Bytes zwischen dem Feld chpbf1 und dem letzten Feld bezeichnet, das nicht den Defaultwert enthält. Als Defaultwert wird dabei für das Feld chpbf1 der Wert 1 (Standard-Zeichenformat), für das Feld hps der Wert 24 (Zeichengröße ist 12 Punkt) und für alle anderen Felder der Wert 0 festgelegt.

Wenn nun zum Beispiel das Feld chpbf3 als letztes Feld innerhalb des FPROP nicht den Defaultwert enthält, müssen nur die Felder chpbf1 bis chpbf3 erfaßt werden. Das Feld cch enthält in diesem Fall den Wert 4.

Die Felder chpbf1 bis fpres innerhalb der Abbildung 5 bezeichnet man übrigens auch als Character Property, abgekürzt CHP. Innerhalb des CHPs spielen verschiedene Bitfelder eine große Rolle. Den Aufbau dieser Bitfelder zeigen die *Abbildungen 6 bis 9.*

Wie die *Abbildung 6* zeigt, gibt das Bit 0 innerhalb des Bitfeldes chpbf1 an, ob das/die beschriebenen Zeichen mit einem Zeichenmuster aus der Druckformatvorlage formatiert wurden. Trifft dies zu, wird der Inhalt der Bits 1 bis 7 als die Varianten-Nummer dieses Zeichenmusters interpretiert (stc: Style Code), wobei 0 für das Standard-Zeichenformat in dem jeweiligen Absatz steht *(Abbildung 7).*

Die möglichen Werte eines Style Codes (stc)		
stc	Verwendung	Variante
0	Zeichen	(reserviert: Standardzeichen)
1-12	Zeichen	1-12
13	Zeichen	Fußnotenverweis
14-18	Zeichen	13-17
19	Zeichen	Seitenzahl
20-27	Zeichen	18-25 (S0-S7 in SCREEN.DFV)
28	Zeichen	Kurzinformationen
29	Zeichen	Zeilennummern
30	Absatz	Standard
31-38	Absatz	1-8
39	Absatz	Fußnotentext
40-87	Absatz	9-56 (56 = SC in SCREEN.DFV)
88-94	Absatz	Überschriften Ebene 1-7
95-98	Absatz	Index Ebenen 1-4
99-102	Absatz	Tabellen Ebenen 1-4
103	Absatz	Kopf-/Fußzeile
105	Bereich	Standard
106-126	Bereich	1-21

Abbildung 7: *Der Style Code (stc) in den Properties.*

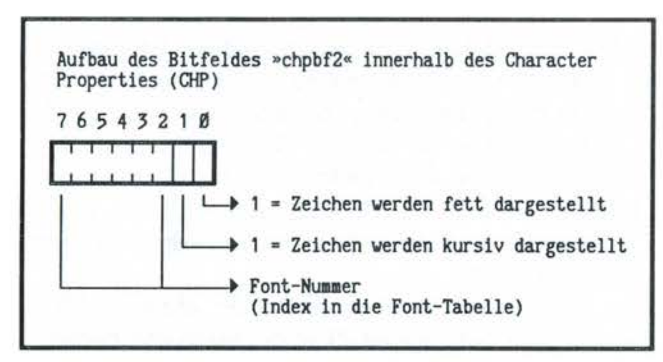


Abbildung 8: *Aufbau des Bitfeldes* chpbf2 *innerhalb des Character Properties (CHP).*

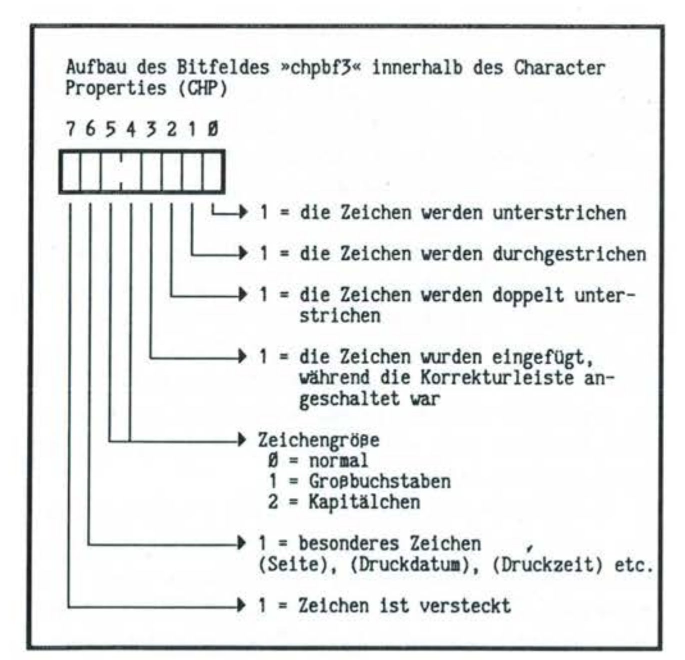


Abbildung 9: *Aufbau des Bitfeldes* chpbf3 *innerhalb des Character Properties (CHP).*

Aufbau des Formatting Property (FPROP) innerhalb einer Page mit Absatzformaten			
Ofs.	Name	Bedeutung	Typ
0	cch	Anzahl der Bytes in diesem FPROP exklusive dieses Bytes	1 BYTE
1	papbf1	Code des Absatzformats aus der Druckformatvorlage*	1 BYTE
2	papbf2	Absatzattribute*	1 BYTE
3	stcNorm	Varianten-Nummer des Standardzeichenformats	1 BYTE
4	papbf3	Absatzattribute*	1 BYTE
5	dxaRight	rechter Absatzeinzug in Twips**	1 WORD
7	dxaLeft	linker Absatzeinzug in Twips	1 WORD
9	dxaLeft1	linker Absatzeinzug der ersten Zeile in Twips	1 WORD
11	dyaLine	Zeilenabstand in Twips	1 WORD
13	dyaBefore	Anfangsabstand in Twips	1 WORD
15	dyaAfter	Endeabstand in Twips	1 WORD
17	papbf4	Absatzattribute*	1 BYTE
18	tyframe	Umrahmung*	1 BYTE
19 -22	papres	reserviert, müssen 0 enthalten	4 BYTE
23 -102	rgTBD	Array mit Tabulatorbeschreibern (TBDs)	20 TBD

*hier handelt es sich um Bitfelder, die auf den folgenden Seiten beschrieben werden
**ein Twip entspricht einem 1/20stel Punkt und damit einem 1/1440stel Zoll

Abbildung 10: *Aufbau des Formatting Properties in einer Page mit Absatzformaten.*

Weichen einzelne Attribute des Zeichenformats von dem Zeichenmuster ab, oder wurde das Zeichen nicht über ein Zeichenmuster aus der Druckformatvorlage formatiert, geben die weiteren Bitfelder Aufschluß über das Format der Zeichen.

Absatzformate

Die Art und Weise, in der MS-Word die Formate der verschiedenen Textabsätze speichert, unterscheidet sich nur wenig von der Speicherung der Zeichenformate. Auch hier nimmt jeweils eine Page die Formate eines oder mehrerer Absätze auf, wobei der Aufbau der Page mit der bei Zeichenformaten identisch ist. Auch hier gilt, daß die Absätze beginnend mit dem ersten Zeichen des Textes in der ersten Absatzformat-Page, fortlaufend beschrieben werden, bis der Absatz erreicht ist, in dem sich das letzte Zeichen des Textes befindet.

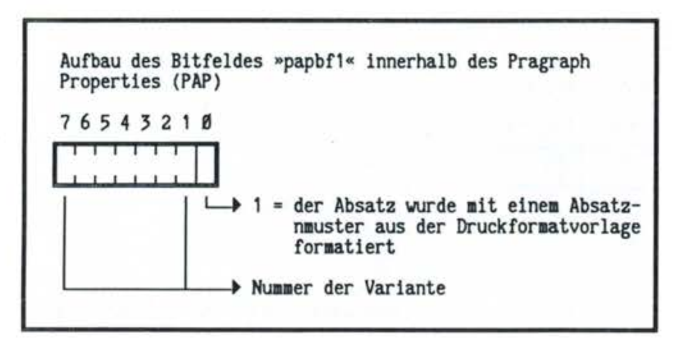


Abbildung 11*: Aufbau des Bitfeldes* `papbf1` *innerhalb des Paragraph Properties (PAP).*

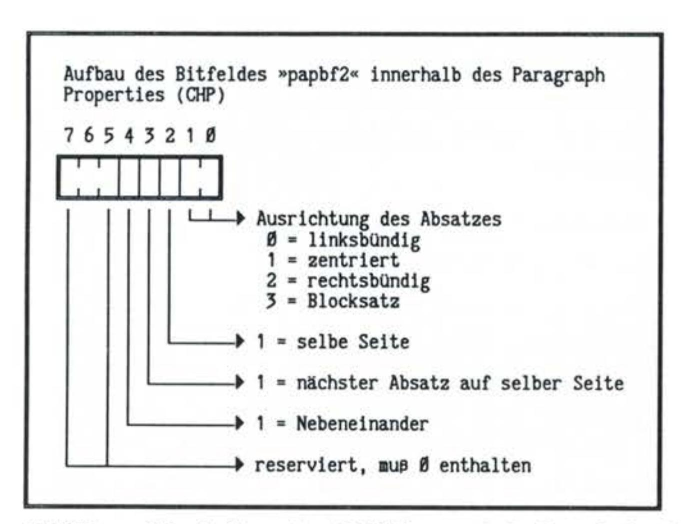


Abbildung 12: *Aufbau des Bitfeldes* `papbf2` *innerhalb des Paragraph Properties (PAP).*

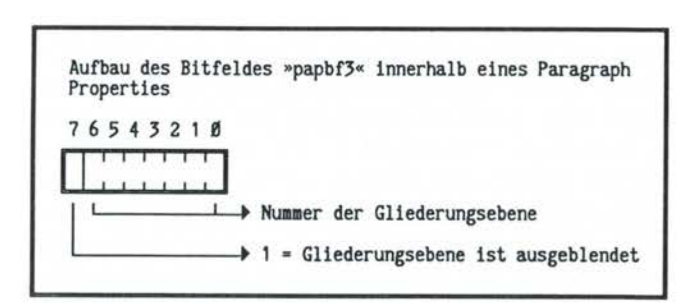


Abbildung 13: *Aufbau des Bitfeldes* `papbf3` *innerhalb eines Paragraph Properties (PAP).*

Anstelle des Zeichenformats einer Gruppe von Zeichen beschreibt der Formatting Property hier das Format des Absatzes, in dem sich die jeweiligen Zeichen befinden. Der einzige Unterschied zu den Zeichenformaten besteht deshalb auch in dem Aufbau dieser Datenstruktur, die in der *Abbildung 10* beschrieben wird.

Die Nummer der ersten Page mit Absatzformaten gibt das Feld `PnFntb` im Vorspann der Datei an.

Um einen `FPROP` mit Absatzformaten auch sprachlich von einem solchen mit Zeichenformaten abgrenzen zu können, werden die Felder `papbf1` bis `rgTBD` als `Paragraph`

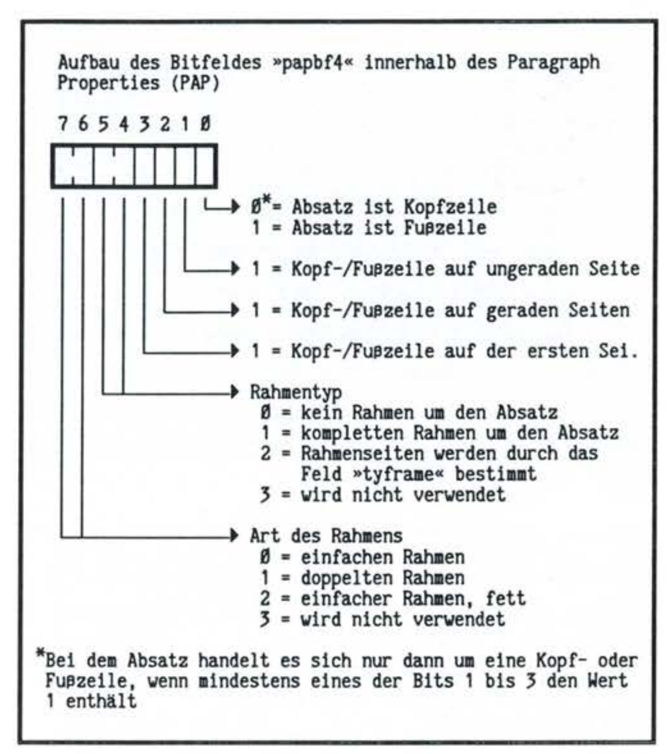


Abbildung 14*: Aufbau des Bitfeldes* `papbf4` *innerhalb des Paragraph Properties (PAP)*

`Property`, kurz `PAP`, bezeichnet. Auch für den `PAP` gilt, daß MS-Word versucht, möglichst wenige Felder abzuspeichern, um Speicherplatz zu sparen. Die Anzahl der gespeicherten Bytes wird wiederum innerhalb des Feldes `cch` erfaßt. Abhängig ist dies auch hier davon, inwieweit der Inhalt der Felder dem von MS-Word festgelegten Defaultwerten entspricht.

Der Defaultwert für das Feld `papbf1` ist 61 (Standardabsatz), für das Feld `stcNorm` 30 (Font), für das Feld `dyaLine` 240 (12 Punkte) und für alle anderen Felder 0.

Ob der jeweilige Absatz über ein Absatzmuster innerhalb der Druckformatvorlage oder ausschließlich »von Hand« formatiert wurde, zeigt das Bitfeld `papbf1` an (*Abbildung 11*). Die Absatzvarianten wurden bereits in Abbildung 7 beschrieben.

Wird innerhalb des Textes mit Gliederungsebenen gearbeitet, spielt das Bitfeld `papbf3` innerhalb des Paragraph Properties eine wichtige Rolle (*Abbildung 13*).

Ob es sich bei dem Absatz um eine Kopf- oder Fußzeile handelt, und ob der Absatz mit einem Rahmen versehen werden soll, darüber gibt das Feld `papbf4` Auskunft (*Abbildung 14*).

Enthalten die Bits 4 und 5 innerhalb des Feldes `papbf4` den Wert 2, dann bestimmt das Feld `tyframe` die Art des Rahmens, d.h. die Seiten, an denen der Rahmen geschlossen ist (*Abbildung 15*).

Ob der Absatz über die Tabulatoren in dem jeweiligen Absatzmuster hinaus mit weiteren Tabulatoren versehen wurde, erkennen Sie am Inhalt des Feldes `cch`.

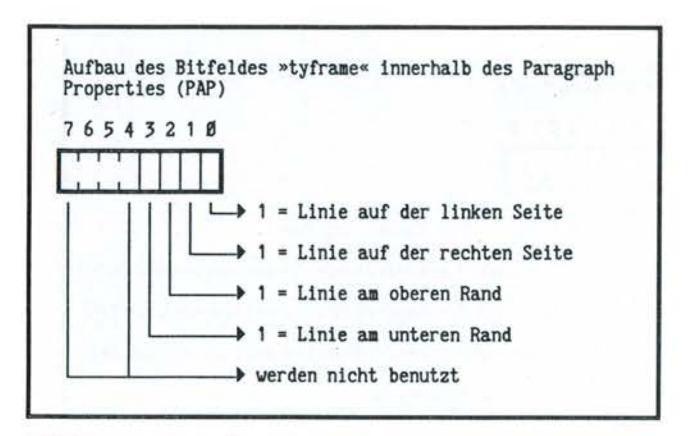


Abbildung 15: *Aufbau des Bitfeldes* `tyframe` *innerhalb des Paragraph Properties (PAP)*

Aufbau eines »Tab Descriptors« (TBD) innerhalb eines PAP

Ofs.	Name	Bedeutung	Typ
Ø	dxa	Abstand des Tabulators vom linken Rand in Twips	1 WORD
2	tbdbf1	Tabulator-Attribute	1 BYTE
3	chAlign	Zeichen, nach dem der Tabulator ausgerichtet werden soll, wenn es sich um einen Dezimaltabulator handelt*	1 BYTE

*befindet sich in diesem Feld der ASCII-Code Ø, wird das Zeichen als "." interpretiert

Abbildung 16: *Aufbau eines Tab Descriptors (TBD) innerhalb eines Paragraph Properties.*

Nur wenn Tabulatoren angelegt wurden, enthält es einen Wert größer 22 und zeigt damit an, daß das Array `rgTBD` Informationen enthält. Da jeder Tab Descriptor (`TBD`) innerhalb des Arrays 4 Bytes beansprucht, läßt sich aus dem Inhalt von `cch` auch gleich die Anzahl der Tabulatoren errechnen:

```
AnzTab = ( cch - 22 ) / 4
```

Der Tab Descriptor hat die in *Abbildung 16* gezeigte Struktur. Die *Abbildung 17* zeigt, wie die Attribute des Tabulators innerhalb des Bitfeldes `tbdbf1` kodiert werden.

Fußnoten

Enthält ein Text Fußnoten, muß MS-Word natürlich auch Informationen über die Fußnotenverweise und die zugehörigen Fußnotentexte speichern. Sowohl die Verweise, als auch die Texte selbst sind dabei ganz normaler Bestandteil des Textes und werden wie alle anderen Zeichen innerhalb des Textteils der Datei gespeichert. Um sich jedoch die Position der Verweise und der zugehörigen Texte merken zu können, legt MS-Word eine sogenannte *Footnote Table* an, die in ihrer Kurzbezeichnung als `FNTB` bezeichnet wird.

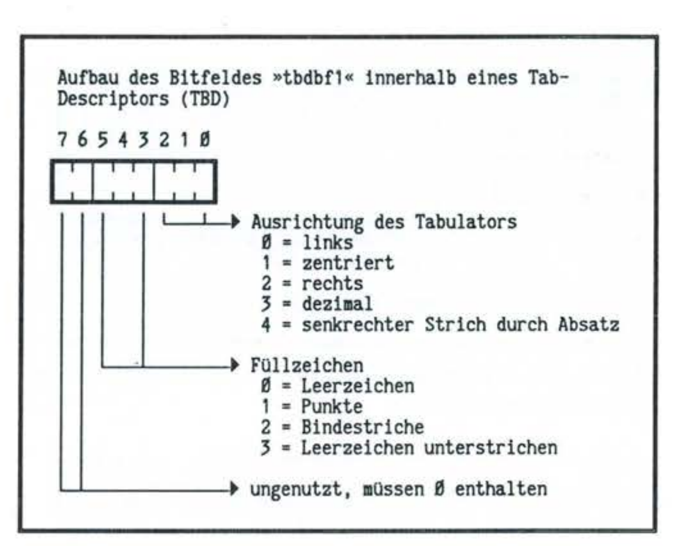


Abbildung 17: *Aufbau des Feldes* `tbdbf1` *innerhalb eines Tab Descriptors.*

Aufbau der »Footnote Table« (FNTB)

Ofs.	Name	Bedeutung	Typ
Ø	cfnd	Anzahl der Fußnoten + 1	1 WORD
2	cfndMax	identisch mit »cfnd«	1 WORD
4 -...	rgfnd	Array mit den einzelnen Fußnotenbeschreibern (FNDs)	x * FND

Abbildung 18: *Aufbau der Footnote Table (FNTB).*

Ob sich eine derartige Tabelle innerhalb einer MS-Word-Datei befindet, läßt sich anhand des Feldes `pnFntb` im Vorspann der Datei feststellen. Enthält dieses Feld den gleichen Wert wie das darauffolgende Feld `pnSep`, enthält die Datei keine Fußnotentabelle und der Text damit auch keine Fußnoten. Unterscheiden sich die beiden Werte jedoch, gibt `pnFntb` die Nummer der Page an, mit der die Fußnotentabelle beginnt. Die *Abbildung 18* zeigt den Aufbau dieser Tabelle. Die Größe der `Footnote Table` und damit die Anzahl der Pages, die durch sie belegt werden, ergibt sich aus der Anzahl der Fußnoten und aus der Größe eines *Footnote Descriptors* (`FND`) nach folgender Formel:

```
fnPages = ( 4 + 8 * ( cfnd - 1 ) ) / 128
```

Wird mehr als eine Page für die Footnote Table benötigt, setzt sich das Array `rgfnd` nahtlos in den folgenden Pages fort, wobei die letzte Page bis zu ihrem Ende mit Nullen aufgefüllt wird. Die einzelnen Fußnotenbeschreiber (Footnote Descriptors, `FNDs`) sind dabei so aufgebaut, wie das in *Abbildung 19* zu sehen ist.

Beachten Sie, daß die einzelnen `FNDs` innerhalb der Footnote Table sortiert nach dem Feld `cpRef` in aufsteigender Reihenfolge aufgeführt werden und daß die Offsetpositionen in den Feldern `cpRef` und `cpFtn` im Gegensatz zu den Positionsangaben in den Zeichen- und Paragraphenformaten nicht ab 128, sondern ab 0 gezählt werden.

Aufbau eines »Footnote Descriptors« (FND)			
Ofs.	Name	Bedeutung	Typ
Ø	cpRef	Offsetposition des Fußnotenverweises im Text	1 DWORD
4	cpFtn	Offsetposition des Fußnotentextes im Text	1 DWORD

Abbildung 19: *Aufbau eines Footnote Descriptors (FND).*

Aufbau der »Section Table« (SETB)			
Ofs.	Name	Bedeutung	Typ
Ø	csed	Anzahl der Bereiche (mindestens 1)	1 WORD
2	csedMax	entspricht dem Feld »csed«	1 WORD
4 -...	rgsed	Array mit Segmentbeschreibern	X * SED

Abbildung 20: *Aufbau der Section Table (SETB).*

Bereichsformate

Untergliedert sich ein Text in verschiedene Bereiche oder wird der einzige Bereich innerhalb eines Textes »von Hand« formatiert und weist dadurch nicht mehr das Standard-Bereichsformat auf, legt MS-Word für jeden Bereich einen Bereichsbeschreiber (`Section Descriptor`, `SED`) an, den es in die `Section Table` (`SETB`) eingliedert. Ob eine solche Tabelle existiert und in welcher Page sie beginnt, erkennen Sie am Inhalt des Feldes `pnSetb` innerhalb des Vorspanns. Ist sein Inhalt mit dem des darauffolgenden Feldes `pnPgtb` identisch, dann enthält die Datei keine Bereichsbeschreiber. Anderenfalls gibt der Inhalt von `pnSetb` die Nummer der Page an, mit der die *Section Table* beginnt.

Aus der Anzahl der Bereiche (Feld `csed`) ergibt sich auch die Anzahl der Pages, über die sich das Array `rgsed` erstreckt, sofern es zu groß ist, um in einer Page Platz zu finden. Die folgende Formel gibt die Anzahl der belegten Pages an und berücksichtigt dabei die Größe der Segmentbeschreiber.

```
sePages = ( 4 + ( csed * 1Ø ) ) / 128
```

Da die Section Table in den seltensten Fällen genau mit dem Ende einer Page abschließt, wird der ungenutzte Teil der letzten Page mit Nullen gefüllt. Die einzelnen Segmentbeschreiber innerhalb des Arrays `rgsed` haben die in *Abbildung 21* beschriebene Struktur.

Über das Feld `cp` innerhalb des `SED`s werden die einzelnen Bereiche innerhalb des Textes voneinander abgegrenzt. Der erste `SED` beschreibt dabei den Bereich vom ersten Zeichen des Textes bis zu dem Zeichen, auf das sein `cp`-Feld zeigt.

Aufbau eines »Segment Descriptors« (SEDs) innerhalb der »Section Table« (SETB)			
Ofs.	Name	Bedeutung	Typ
Ø	cp	Offsetadresse des Absatz-Ende-Zeichens innerhalb des Textes	1 DWORD
4	fn	wird nicht genutzt	1 WORD
6	fcSep	Offsetadresse des zugehörigen »Segment Properties« innerhalb der Datei	1 DWORD

Abbildung 21: *Aufbau eines Segment Descriptors (SED).*

Aufbau eines »Section Properties« (SEP)			
Ofs.	Name	Bedeutung	Typ
Ø	cch	Anzahl der Bytes innerhalb des SEPs exklusive dieses Bytes	1 BYTE
1	sepbf1	Bereichsmuster*	1 BYTE
2	sepbf2	Bereichs-Attribute*	1 BYTE
3	yaMac	Seitenlänge in Twips	1 WORD
5	xaMAc	Seitenbreite in Twips	1 WORD
7	pgnStart	erste Seitennummer	1 WORD
9	yaTop	oberer Rand in Twips	1 WORD
11	dyaText	Länge des Textbereichs in Twips	1 WORD
13	xaLeft	linker Rand in Twips	1 WORD
15	dxaText	Breite des Textbereichs in Twips	1 WORD
17	sepbf3	weitere Absatzattribute*	1 BYTE
18	cColumns	Anzahl der Spalten	1 BYTE
19	yaRH1	Abstand der Kopfzeile vom Blattanfang in Twips	1 WORD
21	yaRH2	Abstand der Fußzeile vom Blattende in Twips	1 WORD
23	dxaCol	Abstand zwischen den Spalten in Twips	1 WORD
25	dxaGutter	Breite des Bundstegs in Twips	1 WORD
27	yaPgn	Abstand der Seitennummer vom oberen Blattrand in Twips	1 WORD
29	xaPgn	Abstand der Seitennummer vom linken Blattrand in Twips	1 WORD
31	dxaLnn	Abstand der Zeilennummern vom linken Blattrand in Twips	1 WORD
33	dLnn	gibt an, jede wievielte Zeile mit einer Zeilennummer versehen werden soll	1 WORD
*es handelt sich bei diesem Feld um ein Bitfeld, das auf den folgenden Seiten beschrieben wird			

Abbildung 22: *Aufbau eines Section Properties (SEP).*

Defaultwerte für die einzelnen Felder eines »Section Properties« (SEP)

Name	Defaultwert	Name	Defaultwert
cch	variabel	sepbf3	0
sepbf1	0	cColumns	1
sepbf2	0	yaRH1	720
yaMac	15840	yaRH2	10440
xaMAc	12240	dxaCol	720
pgnStart	1	dxaGutter	0
yaTop	1440	yaPgn	720
dyaText	12960	xaPgn	10440
xaLeft	1800	dxaLnn	576
dxaText	8640	dLnn	1

Abbildung 23: *Defaultwerte für die einzelnen Felder eines Section Properties (SEP).*

Der nachfolgende SED beschreibt dann den Bereich zwischen dem cp-Feld seines Vorgängers plus 1 und seinem eigenen cp-Feld. In dieser Weise beschreiben die einzelnen SEDs die verschiedenen Bereiche innerhalb des Textes in aufsteigender Reihenfolge, so daß das cp-Feld des letzten SEDs hinter das letzte Zeichen im Text zeigt. Beachten Sie dabei aber bitte, daß die Zeichenpositionen innerhalb der Datei im Gegensatz zu den Zeichen- und Absatzbeschreibern nicht ab 128, sondern ab 0 gezählt werden.

Wie die *Abbildung 21* zeigt, beschreibt der SED das Format und die Attribute »seines« Bereichs jedoch nicht selbst, sondern überläßt diese Aufgabe einem *Section Property* (SEP), dessen Offsetadresse innerhalb der Datei er angibt. Ähnlich wie bei den Zeichen- und Absatzbeschreibern ergibt sich dadurch die Möglichkeit, mehrere Bereiche durch einen SEP beschreiben zu lassen und dadurch Speicherplatz zu sparen. Den Aufbau eines solchen SEPs verdeutlicht die *Abbildung 22*.

Auch bei der Erstellung der SEPs nutzt MS-Word die Möglichkeit, Speicherplatz zu sparen, indem nur der Teil des SEPs gespeichert wird, dessen Inhalt nicht den Defaultwerten entspricht. Wie bei den Character- und Paragraph-Properties gibt auch im SEP ein Feld mit dem Namen cch die Anzahl der gespeicherten Bytes innerhalb der Struktur an. Welche Werte MS-Word bei den einzelnen Feldern als Defaultwerte voraussetzt, zeigt die *Abbildung 23*.

Zwei wichtige Informationen, die Position des rechten und des unteren Endes des bedruckbaren Blattbereichs, gehen aus dem SEP nicht direkt hervor, können aber durch eine Verknüpfung der anderen Felder errechnet werden. Die entsprechenden Formeln lauten:

```
xaRight = xaMac - xaLeft - dxaText
yaBottom = yaMac - yaTop - dyaText
```

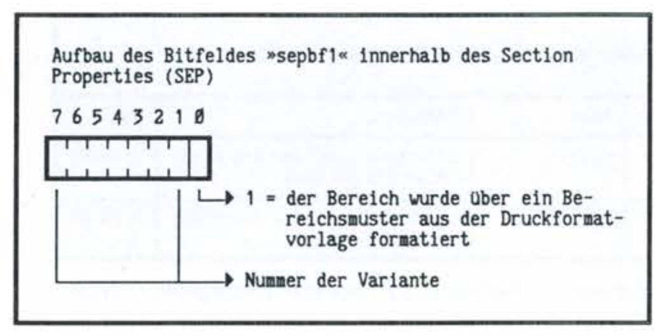


Abbildung 24: *Aufbau des Bitfeldes* sepbf1 *innerhalb eines Section Properties (SEP).*

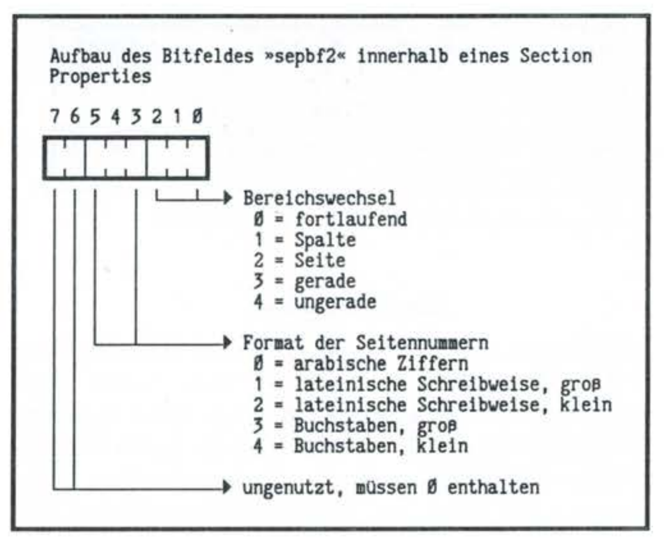


Abbildung 25: *Aufbau des Bitfeldes* sepbf2 *innerhalb eines Section Properties (SEP).*

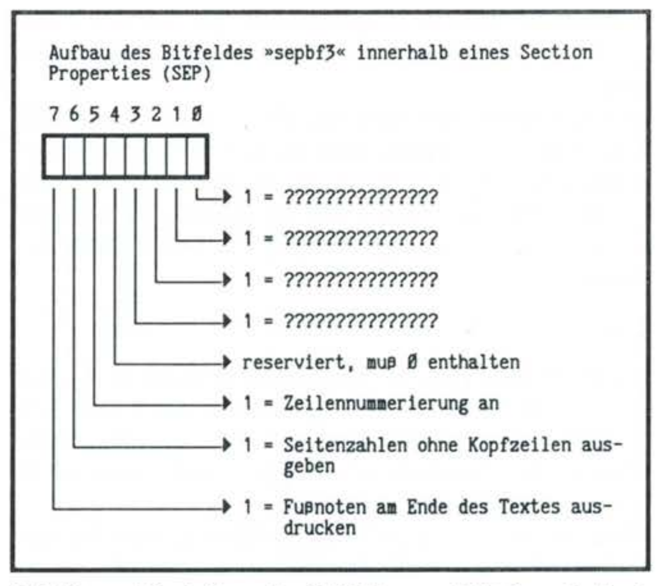


Abbildung 26: *Aufbau des Bitfeldes* sepbf3 *innerhalb eines Section Properties (SEP).*

Aufbau der »Page Table« (PGTB)			
Ofs.	Name	Bedeutung	Typ
0	cpgd	Anzahl der Druckseiten	1 WORD
2	cpgdMac	ungenutzt	1 WORD
4 -...	rgpgd	Array mit den einzelnen Seitenbeschreibern (PGDs)	x * PGD

Abbildung 27: *Aufbau der Page Table (PGTB).*

Aufbau eines »Page Descriptors« (PGD)			
Ofs.	Name	Bedeutung	Typ
0	pgn	Nummer der Druckseite	1 WORD
4	cpMin	Offsetposition des ersten Zeichens dieser Druckseite innerhalb des Textes	1 DWORD

Abbildung 28: *Aufbau eines Page Descriptors (PGD).*

Wie die Abbildung 22 zeigt, enthält ein Section Property verschiedene Felder, deren einzelne Bits Informationen über die Attribute des Bereichs liefern. Das erste dieser Felder trägt den Namen `sepbf1` und zeigt an, ob der Bereich über ein Bereichsmuster aus der Druckformatvorlage formatiert wurden (*Abbildung 24*). Die Varianten der Bereichsmuster wurden bereits in Abbildung 7 beschrieben.

Über den Bereichswechsel und das Format der Seitennummern gibt das Feld `sepbf2` Auskunft (*Abbildung 25*).

Position der Seitenumbrüche

Nach dem ersten Druck eines Textes kennt MS-Word die Lage der Seitenumbrüche innerhalb des Textes und speichert diese Information innerhalb der Textdatei hinter der letzten `SEP`-Page ab.

Ob der Text bereits mindestens einmal gedruckt wurde und diese Informationen damit zur Verfügung stehen, erkennt man am Inhalt des Feldes `pnPgtb` im Vorspann der Datei. Ist sein Inhalt mit dem des darauffolgenden Feldes (`pnSumd`) identisch, liegen diese Informationen noch nicht vor. Anderenfalls gibt der Inhalt von `pnPgtb` die Nummer der Page innerhalb der Datei an, mit der die Informationen über die Lage der Seitenumbrüche beginnen. Diese Page enthält die sogenannte *Page Table*, deren Aufbau die *Abbildung 27* zeigt.

Die Größe der `Page Table` und damit die Anzahl der (Disk-) Pages, die durch sie belegt werden, ergibt sich aus der Anzahl der Druckseiten und aus der Größe eines Page Descriptors (`PGD`) nach folgender Formel:

```
pgPages = ( 4 + 6 * ( cpgd - 1 ) ) / 128
```

Wird mehr als eine Page für die `Page Table` benötigt, setzt sich das Array `rgpgd` nahtlos in den folgenden Pages fort, wobei die letzte Page bis zu ihrem Ende mit Nullen aufgefüllt wird. Die einzelnen Seitenbescheiber (Page Descriptors, `PGDs`) sind dabei so aufgebaut, wie es in *Abbildung 28* zu sehen ist.

Aufbau der Page mit Informationen über den Text			
Ofs.	Name	Bedeutung	Typ
0	oTitle	Titel*	1 WORD
2	oAuthor	Autor*	1 WORD
4	oOperator	Sachbearbeiter*	1 WORD
6	oKeywords	Kennwort*	1 WORD
8	oComments	Kommentare*	1 WORD
10	oVersion	Versionsnummer des Textes*	1 WORD
12	dRevision	Datum der letzten Überarbeitung**	1 WORD
14	dCreation	Datum der Erstellung**	1 WORD
16	nChars	Anzahl der Zeichen im Text	1 DWORD

*diese Felder enthalten die Offsetposition der jeweiligen Information innerhalb der Page. Die Information selbst ist dort im Format eines C-Strings verzeichnet.

**diese Felder enthalten die Offsetposition des Datums innerhalb der Page. Das Datum ist jeweils als ein Feld bestehend aus 8 Zeichen aufgebaut.

Abbildung 29: *Aufbau der Page mit Informationen über den Text.*

Informationen über den Text

MS-Word bietet ab der Version 4.0 die Möglichkeit, neben dem eigentlichen Text auch Informationen über den Text, wie etwa den Titel, den Namen des Autors und einen Kommentar zu erfassen. Diese Informationen speichert es in einer gesonderten Page innerhalb der Textdatei ab, deren Page-Nummer es innerhalb des Vorspanns in dem Feld `pnSumd` verzeichnet. Den Aufbau dieser Page zeigt die *Abbildung 29.*

Das Programm BW.C

Ein schönes Beispiel für die Erstellung von MS-Word-Dateien ist das Programm BW.C, dessen Listing Sie auf den folgenden Seiten finden. Es schließt an das TSR-Programm BILD an, das wir in der November/Dezember-Ausgabe des letzten Jahres veröffentlicht haben.

Nachdem Sie mit Hilfe von BILD die Hardcopy eines 80*25-Zeichen-Textbildschirms in eine Datei geschrieben haben, können Sie sie mit Hilfe von BW (das steht für »BildWord«) in eine MS-Word-Datei umwandeln lassen, die Sie wie jede andere Text-Datei in Ihre Texte einbinden können. Dadurch wird es möglich, Hardcopies als integralen Bestandteil eines Textes zu behandeln und dadurch den

Aufwand des »Einmontierens« zu ersparen. Außerdem hilft Ihnen die direkte Einbindung der Hardcopies natürlich auch bei der Formatierung des Textes, da MS-Word die Größe der Hardcopy erkennt.

BW ist jedoch nicht von BILD abhängig. Ihm kommt es lediglich darauf an, eine Datei vorzufinden, die die 4000 Bytes eines 80*25-Zeichen-Textbildschirms in der Reihenfolge enthält, wie sie innerhalb des Video-RAMs dargestellt werden. Wenn Sie also mit einem anderen Hardcopy-Programm arbeiten möchten, steht dem grundsätzlich nichts im Wege.

Aufbau von BW

Das BW-Programm setzt sich aus zwei Dateien zusammen: die Include-Datei WORD.H (*Listing 1*) und das eigentliche Programm in der Datei BW.C (*Listing 2*). In dieser Datei finden Sie eine Deklaration aller Strukturen innerhalb einer MS-Word-Datei, wie sie auf den vorhergehenden Seiten beschrieben wurden. Ebenfalls finden Sie dort die Deklaration der verschiedenen Bit-Felder, die die einzelnen Strukturen beinhalten. Sowohl die Strukturen als auch die Bitfelder werden über den `typedef`-Befehl als eigenständige Datenstrukturen deklariert, wobei ihr Name mit dem Namen des korrespondierenden Objekts aus der MS-Word-Datei übereinstimmt. Diese Include-Datei eignet sich daher ideal als Basis zur Erstellung eigener Programme, die MS-Word-Dateien lesen, manipulieren oder erstellen.

Eng verbunden mit dem BW-Programm ist auch die Druckformatvorlage SCREEN.DFV (*Bild 1*), die die Varianten zur Formatierung der Hardcopy enthält. Da die erstellte Textdatei speziell auf diese Druckformatvorlage ausgerichtet ist und ihr Name innerhalb des Vorspanns der Textdatei angegeben wird, sollten Sie diese Druckformatvorlage mit der Druckformatvorlage Ihres jeweiligen Textes verbinden, sofern Sie Hardcopies in den Text einbinden möchten. Der Befehl von MS-Word dazu lautet: `Muster Übertragen Zusammenführen`.

Die Hardcopy selbst wird als ein Absatz erstellt, der mit der Absatzvariante `SC` formatiert wird. Die einzelnen Zeichen werden je nach ihrer Farbe mit unterschiedlichen Attributen versehen. Inverse Zeichen werden kursiv dargestellt, helle Zeichen durch das Attribut »fett« hervorgehoben und unterstrichene Zeichen im Ausdruck ebenfalls unterstrichen.

Einige Zeichen müssen bei der Umwandlung in eine MS-Word-Datei umgesetzt werden, da Sie von MS-Word in besonderer Weise interpretiert werden und deshalb nicht im Text auftauchen sollten. Dies gilt für das Zeichen mit dem Code 9 (Tabulator) und für das Zeichen mit dem Code 31, das ebenfalls umgesetzt werden muß, da es von MS-Word sonst als bedingter Trennstrich betrachtet wird.

Über die einfache Umwandlung der Hardcopy hinaus bietet BW die Möglichkeit, die Hardcopy mit einem einfachen oder einem doppelten Rahmen zu umgeben. Ersteres wird als Defaulteinstellung vorgegeben.

```
/*
[ ]-----------------------------------------------------[ ]
 |                                                       |
 |     Include-Datei     : WORD.H                        |
 |     zum Zugriff       : auf die Strukturen innerhalb  |
 |                         einer MS-Word-Datei           |
 |     erstellt am       : 12.02.1988                    |
 |     letztes Update am: 12.02.1988                     |
 |                                                       |
 |     (Copyright)       : 1989 by MICHAEL TISCHER       |
 |                         and GÜNTER JÜRGENSMEIER       |
 |                                                       |
[ ]-----------------------------------------------------[ ]
*/

/*-- Kompilierungs-Befehle ---------------------------------*/

#pragma pack(1)                 /* Strukturen byteweise packen */

/*== Konstanten ============================================*/

#define PAGELEN 128  /* Länge einer Page innerhalb der Datei */

/*== skalare Typedefs ======================================*/

typedef unsigned char BYTE;        /* wir basteln uns ein BYTE */
typedef unsigned int WORD;                     /* dito: WORD */
typedef unsigned long ULONG;                    /* ein DWORD */
typedef char near * CNP;           /* NEAR-Pointer auf Char */

/*== Typedefs der Bitfelder ================================*/

typedef struct   /* Bit-Feld für bvRev im Vorspann der Datei */
{
 BYTE fMarkRev : 1,        /* Veränderungen merken, Ja/Nein   */
      RevText  : 3,        /* Attribut für veränderte Zeichen */
      RevBar   : 2,        /* Position der Korrekturleiste    */
      Dummy    : 2;        /* ungenutzt                       */
} BFREV;

typedef struct                 /* Bit-Feld für CHPBF1 aus FPROPC */
{
 BYTE fStyled : 1,             /* 1 = mit Muster formatiert    */
      stc     : 7;             /* Nummer der Zeichen-Variante  */
} CHPBF1;

typedef struct                 /* Bit-Feld für CHPBF2 aus FPROPC */
{
 BYTE fBold   : 1,                  /* 1 = Zeichen ist fett   */
      fItalic : 1,                  /* 1 = Zeichen ist kursiv */
      ftc     : 6;                  /* Font-Nummer            */
} CHPBF2;

typedef struct                 /* Bit-Feld für CHPBF3 aus FPROPC */
{
 BYTE fUline   : 1,     /* 1 = Zeichen ist unterstrichen      */
      fStrike  : 1,     /* 1 = Zeichen ist durchgestrichen    */
      fDline   : 1,     /* 1 = Zeichen ist doppelt-unterstr.  */
      fNew     : 1,     /* 1 = Zeichen wurden eingefügt       */
      csm      : 2,     /* Großschrift, Kapitälchen?          */
      fSpecial : 1,     /* 1 = Zeichen ist (Seite),...        */
      fHidden  : 1;     /* 1 = Zeichen ist versteckt          */
} CHPBF3;

typedef struct                 /* Bit-Feld für PAPBF1 aus FPROPP */
{
 BYTE fStyled : 1,             /* 1 = mit Muster formatiert    */
      stc     : 7;             /* Nummer der Absatz-Variante   */
} PAPBF1;

typedef struct                 /* Bit-Feld für PAPBF2 aus FPROPP */
{
 BYTE jc        : 2,    /* Ausrichtung des Absatz             */
      fKeep     : 1,    /* 1 = Absatz selbe Seite             */
      fKFollow  : 1,    /* 1 = folgender Absatz selbst Seite  */
      fSBS      : 1,    /* 1 = nebeneinander                  */
      dummy     : 3;    /* reserviert, enthalten 0            */
} PAPBF2;
```

Listing 1: `WORD.H`

```
typedef struct                   /* Bit-Feld für PAPBF3 aus FPROPP */
{
 BYTE level   : 7,       /* Nummer der Gliederungsebene         */
      fHidden : 1;       /* 1 = Gliederungsebene ausgeblendet */
} PAPBF3;

typedef struct                   /* Bit-Feld für PAPBF4 aus FPROPP */
{
 BYTE fBottom : 1,                 /* 1 = Kopf-, 0 = Fußzeile  */
      fOdd    : 1,                 /* 1 = auf ungeraden Seiten */
      fEven   : 1,                 /* 1 = auf geraden Seiten   */
      fFirst  : 1,                 /* 1 = auf der ersten Seite */
      btc     : 2,                 /* Rahmentyp                */
      bsc     : 2;                 /* Rahmenattribute          */
} PAPBF4;

typedef struct                  /* Bit-Feld für TYFRAME aus FPROPP */
{
 BYTE fBorderLeft  : 1,            /* 1 = Linie links          */
      fBorderRight : 1,            /* 1 = Linie rechts         */
      fBorderAbove : 1,            /* 1 = Linie oben           */
      fBorderBelow : 1,            /* 1 = Linie unten          */
      dummy        : 4;            /* reserviert, enthalten 0  */
} TYFRAME;

typedef struct                     /* Bit-Feld für TBDBF1 aus TBD */
{
 BYTE jcTab : 3,                /* Ausrichtung des Tabulators */
      tlc   : 3,                /* Füllzeichen                */
      dummy : 2;                /* reserviert, enthalten 0    */
} TBDBF1;

typedef struct                     /* Bit-Feld für SEPBF1 aus SEP */
{
 BYTE fStyled : 1,           /* 1 = mit Muster formatiert     */
      stc     : 7;           /* Nummer der Bereichs-Variante */
} SEPBF1;

typedef struct                     /* Bit-Feld für SEPBF2 aus SEP */
{
 BYTE bkc    : 3,                  /* Bereichswechsel          */
      nfcPgn : 3,                  /* Format der Seitennummer */
      dummy  : 2;                  /* reserviert, enthalten 0 */
} SEPBF2;

typedef struct                     /* Bit-Feld für SEPBF3 aus SEP */
{
 BYTE rhc      : 4,      /* reserviert, enthält 0               */
      dummy    : 1,      /* reserviert, enthält 0               */
      fLnn     : 1,      /* 1 = Zeilennummern drucken           */
      fEndFtns : 1;      /* 1 = Fußnoten am Ende des Textes    */
} SEPBF3;

/*== konstante Codes für die verschiedenen Bitfelder ========*/

/*------------------- Codes für RevText in BFREV (VORSPANN) */
#define REV_TEXT_UNDERLINE 0                 /* unterstrichen */
#define REV_TEXT_UPPERCASE 1                /* Großbuchstaben */
#define REV_TEXT_NORMAL    2    /* kein besonderes Attribut */
#define REV_TEXT_BOLD      3                          /* fett */
#define REV_TEXT_DUP_UNDER 6       /* doppelt-unterstrichen */

/*-------------------- Codes für RevBar in BFREV (VORSPANN) */
#define REV_BAR_NO        0      /* keine Korrekturleiste  */
#define REV_BAR_LEFT      1                /* Leiste links */
#define REV_BAR_RIGHT     2               /* Leiste rechts */
#define REV_BAR_ALTERNATE 3   /* abwechselnd links rechts */

/*------------------------ Codes für csm in CHPBF3 (FPROPC) */
#define CSM_NORMAL      0               /* normale Schrift */
#define CSM_UPPER       1                /* Großbuchstaben */
#define CSM_SMALL_CAPS 2                    /* Kapitälchen */

/*------------------------- Codes für jc in PAPBF2 (FPROPP) */
#define JC_LEFT   0                       /* linksbündig  */
#define JC_CENTER 1                       /* rechtsbündig */
#define JC_RIGHT  2                       /* zentriert    */
#define JC_BOTH   3                       /* Blocksatz    */
```

Listing 1: *(Fortsetzung)*

Wird die Hardcopy nicht mit einem Rahmen umgeben, werden am linken und rechten Rand jeder Zeile ein hartes Leerzeichen (ASCII-Code 255) erzeugt. Dies soll dazu beitragen, daß die Hardcopy zentriert werden kann, ohne daß MS-Word die verschiedenen Zeilen unterschiedlich einrückt, weil sie mit weichen Leerzeichen (ASCII-Code 32) beginnen oder enden.

Darüber hinaus erlaubt BW die Erstellung einer Hardcopy zur Weiterverarbeitung durch das Schriftenpaket DocuJet, daß unter anderem den Ausdruck solcher Hardcopies inklusive der Bildschirmattribute auf dem HP-LaserJet ermöglicht. Beim HP-LaserJet gibt es jedoch die Besonderheit, daß sich das Leerzeichen eines Fonts nicht umdefinieren läßt. Deshalb wird ein Leerzeichen von BildWord in den Code 127 umgesetzt, der von DocuJet zum Beispiel als reverses Leerzeichen definiert ist. Übrigens werden alle Ausgaben des *Microsoft System Journals* mit Microsoft Word und DocuJet bearbeitet. Alle Bildschirmabbildungen, die sie in dieser Ausgabe finden, sind mit den Programmen *Bild* und *BildWord* erstellt worden.

Die einzelnen Optionen werden durch die Angabe spezieller Kommandozeilen-Parameter beim Aufruf von BW aktiviert. Hier eine Liste der akzeptierten Parameter, die wahlweise in Groß- oder Kleinschreibung angegeben werden können:

Parameter	Abkürzung	Bedeutung
`/KEINRAHMEN`	`/K`	kein Rahmen um die Hardcopy
`/RAHMEN`	`/R`	einfachen Rahmen um die Hardcopy
`/DOPPELT`	`/2`	doppelten Rahmen um die Hardcopy
`/DOCUJET`	`/D`	Ausgabe für DocuJet

Die Namen der umzuwandelnden Dateien werden ebenfalls in der Befehlszeile angegeben. Sie können beliebig viele Dateinamen angeben, die einzelnen Namen dabei auch mit Wildcards (*, ?) versehen, um eine ganze Gruppe von Dateien zu erfassen. Geben Sie dabei keine Dateierweiterung an, geht BW von der Erweiterung `.BLD` aus, die bei BILD-Dateien automatisch vorliegt. Die erstellten Dateien tragen den gleichen Dateinamen wie die Ursprungsdateien, werden jedoch mit der Erweiterung `.TXT` versehen, die sie als MS-Word-Dateien kennzeichnen.

Die Arbeitsweise von BW

Die Arbeitsweise von BW zeigt einige grundlegende Schwierigkeiten auf, die es bei der Erstellung von MS-Word-Dateien zu meistern gilt. Das größte Problem dabei ist, daß die Informationen bei der Verarbeitung der Eingabedaten (hier der Hardcopy-Datei) nicht in der gleichen Reihenfolge anfallen, wie sie innerhalb der MS-Word-Datei gespeichert werden.

Das beginnt bereits beim Vorspann der Datei. Viele der Informationen, die hier verzeichnet werden (wie etwa die Page-Nummer der Absatz- und Bereichs-Formate), ergeben sich erst während der Erstellung der Datei, dann nämlich, wenn der jeweils vorhergehende Teil der Datei abgeschlossen und dadurch festgestellt wurde, wieviele Pages er belegt. Um aber den Text in die Datei schreiben zu können, muß zuerst der Vorspann ausgegeben werden, damit der Text mit der zweiten Page beginnt.

BW behilft sich hier mit einem Trick, indem es zunächst eine Dummy-Page als Vorspann in die Datei schreibt und erst nach der Erstellung des Textes und der Formatinformationen zum Anfang der Datei zurückkehrt und den Dummy-Vorspann durch den richtigen Vorspann überschreibt.

Nicht so einfach ist das Problem zu lösen, daß beim Durchlaufen der einzelnen Zeilen und Spalten der Hardcopy sowohl die auszugebenden Zeichen (also der Text), als gleichzeitig auch die zugehörigen Formatinformationen anfallen. Während erstere direkt in die Textdatei geschrieben werden können, müssen letztere zunächst zwischengespeichert werden, um erst nach der Ausgabe des letzten (Text-) Zeichens in die Datei geschrieben zu werden.

In diesem Zusammenhang kommt der Funktion `StoreFormat()` eine entscheidende Bedeutung zu. Ihr nämlich werden von der Funktion `WriteChAt()` (Ausgabe eines Zeichens mit dem entsprechenden Attribut) die `Formatting Properties` der Zeichen (`FPROPCs`) übergeben, die beim Durchlaufen der Zeilen und Spalten »anfallen«. `StoreFormat()` trägt diese Informationen in eine Page mit Zeichenformaten (`FPAGE`) ein, die später in die Textdatei geschrieben wird. Da nicht vorherzusehen ist, wie viele dieser `FPROPCs` und damit wieviele Pages erstellt werden müssen (das hängt von der Entropie, also der Mischung der verschiedenen Attribute in der Hardcopy-Datei ab), speichert `StoreFormat()` die einzelnen Pages in einem großen Puffer ab, den es über den Heap allokiert.

Immer, wenn eine Page voll ist, vergrößert es den Page-Puffer um eine neue Page und hängt die neue Page an sein Ende an. `StoreFormat()` weist damit den Charakter einer zentralen Verwaltungsinstanz auf. Während dies im Hinblick auf Organistionen und Behörden oft nicht gerade zu einer Steigerung der Effizienz beiträgt, bringt dies hier jedoch Vorteile mit sich. Dazu zählt vor allem, daß `StoreFormat()` jederzeit weiß, welche Formate (in Form der `FPROPCs`) es bereits in einer Page angelegt hat und dadurch feststellen kann, ob der übergebene `FPROPC` bereits in der Page gespeichert wurde. In diesem Fall kann auf die Aufnahme dieses `FPROPCs` in die Page verzichtet werden, und es muß statt dessen nur ein neuer FOD mit einem Pointer auf den bereits existierenden `FPROPC` angelegt werden. Dies trägt dazu bei, den Gesamtumfang der erstellten Datei möglichst klein zu halten und erhöht dadurch vor allem bei großen Dateien die Performance von MS-Word bei der Verarbeitung der Datei.

```
/*------------------------- Codes für btc in PAPBF4 (FPROPP) */
#define BTC_NO_BORDER 0                /* kein Rahmen um Absatz */
#define BTC_BOX       1                   /* kompletter Rahmen */
#define BTC_SELECT    2                /* Rahmentyp in tyframe */

/*------------------------- Codes für bsc in PAPBF4 (FPROPP) */
#define BSC_SINGLE 0                       /* einfacher Rahmen */
#define BSC_DOUBLE 1                       /* doppelter Rahmen */
#define BSC_BOLD   2                  /* einfacher Rahmen fett */

/*------------------------- Codes für jcTab aus TBDBF1 (TBD) */
#define JCTAB_LEFT    0                                /* links */
#define JCTAB_CENTER  1                            /* zentriert */
#define JCTAB_RIGHT   2                               /* rechts */
#define JCTAB_DECIMAL 3                    /* Dezimal-Tabulator */

/*--------------------------- Codes für tlc aus TBDBF1 (TBD) */
#define TLC_WHITE     0                          /* Leerzeichen */
#define TLC_DOTS      1                               /* Punkte */
#define TLC_HYPHENS   2                         /* Bindestriche */
#define TLC_UNDERLINE 3           /* Leerzeichen unterstrichen */

/*--------------------------- Codes für bks aus SEPBF2 (SEP) */
#define BKS_LINE   0                            /* fortlaufend */
#define BKS_COLUMN 1                                 /* spalte */
#define BKS_RECTO  2                                 /* gerade */
#define BKS_VERSO  3                               /* ungerade */


/*------------------------ Codes für nfcPgn aus SEPBF2 (SEP) */
#define NFCPGN_ARABIC    0                 /* arabische Ziffern */
#define NFCPGN_UC_ROMAN  1      /* lateinische Schreibw., groß */
#define NFCPGN_LC_ROMAN  2     /* lateinische Schreibw., klein */
#define NFCPGN_UC_LETTER 3                 /* in Worten, groß */
#define NFCPGN_LC_LETTER 4                /* in Worten, klein */


/*==========================================================*/

typedef struct                 /* Vorspann einer MS-Word-Datei */
{
 WORD    wIdent;                     /* Identitätscode, 48689 */
 WORD    dty;                   /* reserviert, muß 0 enthalten */
 WORD    wTool;                         /* muß 43776 enthalten */
 WORD    cReceipts;             /* reserviert, muß 0 enthalten */
 WORD    cbReceipts;            /* reserviert, muß 0 enthalten */
 WORD    bReceipts;             /* reserviert, muß 0 enthalten */
 WORD    isgMac;                /* reserviert, muß 0 enthalten */
 ULONG   fcMac;                   /* Anzahl der Zeichen + 128 */
 WORD    pnPara;               /* Page-Nummer: Absatz-Formate */
 WORD    pnFntb;             /* Page-Nummer: Fußnoten-Tabelle */
 WORD    pnSep;          /* Page-Nummer: Bereichs-Beschreiber */
 WORD    pnSetb;             /* Page-Nummer: Bereichs-Tabelle */
 WORD    pnPgtb;         /* Page-Nummer: Druckseiten-Tabelle */
 WORD    pnSumd;              /* Page-Nummer: Infos über Text */
 BYTE    szSsht[66];           /* Name der Druckformatvorlage */
 WORD    pnMacWrite;                   /* pnMac für MS-Write */
 BYTE    szprd[8];               /* Name des Druckertreibers */
 WORD    pnMac;                          /* Anzahl der Pages */
 BFREV   bfRev;                        /* Veränderungs-Flag */
 BYTE    dummy[20];                       /* bisher ungenutzt */
} VORSPANN;


typedef struct                     /* Format Descriptor (FOD) */
{
 ULONG   fcLim;          /* erstes nicht mehr beschriebenes Z. */
 WORD    bfprop;              /* Offset zum zugehörigen FPROP */
} FOD;


typedef struct                        /* eine Page mit Zeichen- */
{                                     /* oder Absatzformaten    */
 ULONG   fcFirst;              /* erstes beschriebenes Zeichen */
 FOD     rgfod[20];                        /* maximal 20 FODs */
 BYTE    dummy[3];                     /* zum Auffüllen auf 128 */
 BYTE    cfod;              /* Anzahl der FODs in dieser Page */
} FPAGE;
```

Listing 1: *(Fortsetzung)*

```
typedef struct                        /* Formatting Property mit        */
{                                     /* Character Property (FPROPC) */
 BYTE     cch;          /* Anzahl der Bytes in diesem FPROP-1 */
 CHPBF1   chpbf1;                           /* Zeichenmuster */
 CHPBF2   chpbf2;                      /* Zeichen-Attribute */
 BYTE     hps;             /* Font-Größe in halben Punkten */
 CHPBF3   chpbf3;              /* weitere Zeichen-Attribute */
 BYTE     hpsPitch;           /* reserviert, muß 0 enthalten */
 BYTE     hpsPos;      /* normal/hochgestellt/tiefgestellt */
} FPROPC;

typedef struct                   /* Tabulator Descriptor (TBD) */
{
 WORD     dxa;                        /* Abstand vom linken Rand */
 TBDBF1   tbdbf1;                  /* Attribute des Tabulators */
 char     chAlign;            /* Zeichen für Dezimaltabulator */
} TBD;

typedef struct                        /* Formatting Property mit        */
{                                     /* Paragraph Property (FPROPP) */
 BYTE     cch;          /* Anzahl der Bytes in diesem FPROP-1 */
 PAPBF1   papbf1;                            /* Absatzmuster */
 PAPBF2   papbf2;                       /* Absatz-Attribute */
 BYTE     stcnorm;          /* Zeichenmuster für std. Zeichen */
 PAPBF3   papbf3;              /* weitere Zeichen-Attribute */
 WORD     dxaRight;                        /* rechter Einzug */
 WORD     dxaLeft;                          /* linker Einzug */
 WORD     dxaLeft1;               /* linker Einzug erste Zeile */
 WORD     dyaLine;                         /* Zeilenabstand */
 WORD     dyaBefore;                     /* Anfangsabstand */
 WORD     dyaAfter;                          /* Endeabstand */
 PAPBF4   papbf4;               /* weitere Absatz-Attribute */
 TYFRAME  tyframe;                            /* Rahmentyp */
 BYTE     dummy[4];             /* reserviert, enthalten 0 */
 TBD      rgTBD[1];                     /* maximal 20 TBDs */
} FPROPP;

typedef struct                  /* Footnote Descriptor (FND) */
{
 ULONG    cpRef;            /* Position der Fußnotenrefernz */
 ULONG    cpFtn;            /* Position des Fußnotentextes */
} FND;

typedef struct                      /* Footnote Table (FNTB) */
{
 WORD     cfnd;                      /* Anzahl der Fußnoten */
 WORD     cfndMax;                             /* reserviert */
 FND      rgfnd[1];      /* Array mit Fußnotenbeschreibern */
} FNTB;

typedef struct                     /* Section Property (SEP) */
{
 BYTE     cch;            /* Anzahl der Bytes in diesem SEP-1 */
 BYTE     sepbf1;                          /* Bereichsmuster */
 BYTE     sepbf2;                       /* Bereichsattribute */
 WORD     yaMac;                              /* Seitenlänge */
 WORD     xyMac;                              /* Seitenbreite */
 WORD     pgnStart;                /* Nummer der ersten Seite */
 WORD     yaTop;                               /* oberer Rand */
 WORD     dyaText;                             /* Drucklänge */
 WORD     xaLeft;                              /* linker Rand */
 WORD     dxaText;                            /* Druckbreite */
 BYTE     sepbf3;                      /* weitere Attribute */
 BYTE     cColumns;                      /* Anzahl der Spalten */
 WORD     yaRH1;                     /* Y-Position Kopfzeile */
 WORD     yaRH2;                      /* Y-Position Fußzeile */
 WORD     dxaCol;                          /* Spaltenabstand */
 WORD     dxaGutter;                   /* Breite des Bundstegs */
 WORD     yaPgn;              /* Y-Position der Seitennummer */
 WORD     xaPgn;              /* X-Position der Seitennummer */
 WORD     dxaLnn;      /* Abstand der Zeilennummern von links */
 WORD     dLnn;                   /* Zeilennummern-Intervall */
} SEP;
```

Listing 1: *(Fortsetzung)*

```
typedef struct                    /* Secton Descriptor (SED) */
{
 ULONG    cp;              /* Offsetposition des Bereichs-Endes */
 WORD     fn;                                   /* reserviert */
 ULONG    fcSep;                  /* Offsetposition des SEP */
} SED;

typedef struct                      /* Section Table (SETB) */
{
 WORD     csed;                      /* Anzahl der Bereiche */
 WORD     csedMax;                              /* reserviert */
 SED      rgsed[1];                         /* Array mit SEDs */
} SETB;

typedef struct                      /* Page Descriptor (PGD) */
{
 WORD     pgn;                            /* Nummer der Seite */
 ULONG    cpMin;           /* Offsetadresse des ersten Zeichens */
} PGD;

typedef struct                          /* Page Table (PGTB) */
{
 WORD     cpgd;                      /* Anzahl der Druckseiten */
 WORD     cpgdMac;                              /* reserviert */
 PGD      rgpgd[1];                         /* Array mit PGDs */
} PGTB;

typedef struct                   /* Informationen über den Text */
{
 CNP      oTitle;                         /* Titel des Textes */
 CNP      oAuthor;                                   /* Autor */
 CNP      oOperator;                         /* Sachbearbeiter */
 CNP      oKeywords;                        /* Schlüsselwörter */
 CNP      oComments;                              /* Kommentare */
 CNP      oVersion;                          /* Versionsnummer */
 CNP      dRecision;                    /* Modifikations-Datum */
 CNP      dCreation;                       /* Erstellungs-Datum */
 ULONG    nChars;                /* Anzahl der Zeichen im Text */
} INFOS;
```

Listing 1: *(Ende)*

Jedoch wird `StoreFormat()` nicht genutzt, um jedes Zeichen mit einem eigenen `FPROPC` zu versehen, da wir bei der Beschreibung des MS-Word-Dateiformats gesehen haben, daß ein `FPROPC` immer nur für ein Gruppe von Zeichen mit gleichem Attribut angelegt wird. Die Entscheidung, ob ein neuer `FPROPC` angelegt wird, fällt dabei nicht `StoreFormat()`, sondern die Funktion `WriteChAt()`, die zur Ausgabe eines Zeichens in die Datei aufgerufen wird.

Um eine Veränderung des Zeichenformats erkennen zu können, speichert sie das Format des jeweils letzten Zeichens in der globlaen Variablen `oldstyle` ab. Wird ihr bei ihrem Aufruf dann ein Zeichen übergeben, das mit einem anderen Format versehen werden muß als es in `oldstyle` verzeichnet ist, erkennt sie den Formatwechsel. Sie ruft dann `StoreFormat()` auf, um einen `FPROPC` für die vorangegangenen Zeichen zu speichern.

Da der `FPROPC` dadurch immer erst im nachhinein angelegt wird, ist es übrigens erforderlich, daß `StoreFormat()` unter Umgehung von `WriteChAt()` nach der Bearbeitung aller Zeichen noch einmal aufgerufen wird, um einen `FPROPC` für das Format der letzten Zeichen zu speichern.

```
1=[********1********2********3********4********5********6********7*****
  1   BS Division Standard                       (C) 1989 G.JÜRGENSMEIER
         Page break. Page length 11"; width 8,5". Page # format Arabic. Top
         margin 1"; bottom 1"; left 0"; right 0". Top running head at 0,5".
         Bottom running head at 0,5". Footnotes on same page.
  2   SC Paragraph 56                            SCREEN
         CourierSeriesII (modern a) 12. Flush left, space after 0,5 li (keep
         in one column).
  3   S0 Character 18                            NORMAL
         CourierSeriesII (modern a) 12.
  4   S1 Character 19                            FETT
         CourierSeriesII (modern a) 12 Bold.
  5   S2 Character 20                            UNTER
         CourierSeriesII (modern a) 12 Underlined.
  6   S3 Character 21                            FETT UNTER
         CourierSeriesII (modern a) 12 Bold Underlined.
  7   S4 Character 22                            REV
         CourierSeriesII (modern a) 12 Italic.
  8   S5 Character 23                            FETT REV
         CourierSeriesII (modern a) 12 Bold Italic.
  9   S6 Character 24                            UNTER REV
         CourierSeriesII (modern a) 12 Italic Underlined.
  10  S7 Character 25                            FETT UNTER REV
         CourierSeriesII (modern a) 12 Bold Italic Underlined.
  ♦
                                                               SCREEN.DFV
```

Bild 1: *Die Druckformatvorlage* `SCREEN.DFV`.

In der Art von `StoreFormat()` kann man sich natürlich auch eine Funktion schaffen, die nicht die Zeichen-, sondern die Absatzformate verwaltet, um sie nach der Bearbeitung aller Zeichen auszugeben. Innerhalb von BW ist dies jedoch nicht erforderlich, da der Aufbau der Page mit den Absatzformaten von der Hardcopy und den Attributen der darin gespeicherten Zeichen unabhängig ist. Grundsätzlich besteht die erstellte Datei aus zwei Absätzen: Dem ersten Absatz mit dem Muster `SC`, der die eigentliche Hardcopy und damit alle Zeichen aus der Datei enthält und einen zweiten Absatz, der ihm nachfolgt und leer ist - dadurch lediglich die Endemarke enthält. Dieser Absatz wird mit dem Format für den Standardabsatz versehen.

Darüber hinaus schreibt BW keine weiteren Informationen in die Datei. Eine Fußnoten-, Bereichs-, oder Seitenumbruchstabelle wird also nicht angelegt, wie man auch nach Bereichsbeschreibern und der Inhaltsangabe vergeblich suchen wird. Wenn Sie noch weitere Fragen zu den Abläufen innerhalb von BW haben, möchten wir Sie an das Listing dieses Programms verweisen. Es ist vollständig dokumentiert und sollte Ihnen deshalb kein Geheimnis verbergen können. Wie Sie das Programm erstellen können, entnehmen Sie bitte dem Kopf der Datei BW.C.

Beachten Sie bei der Erstellung des Programms, daß das Modul `setargv.obj` beim Linkvorgang mit BW verbunden werden muß. Seine Aufgabe ist es, noch vor dem Aufruf der Funktion `main()` und damit vor dem Start des Programms alle Dateien zu ermitteln, die auf die in der Kommandozeile genannten Namensmuster (zum Beispiel `*.BLD`) passen und diese Namen innerhalb des Vektors `argv` zu übergeben. Dem Programm erwächst dadurch der Eindruck, als seien die Namen dieser Dateien in der Kommandozeile angegeben worden und es muß diesen Vorgang nicht in eigener Regie durchführen.

Für Windows-Entwickler sei hier noch angemerkt, daß das Dateiformat von Windows Write in den wesentlichen Punkten mit dem von Word übereinstimmt.

Günter Jürgensmeier / Michael Tischer

```
/**********************************************************************/
/*                    B I L D W O R D   ( B W . C )                   */
/*--------------------------------------------------------------------*/
/*     Aufgabe        : Konvertiert Hardcopy-Dateien, die             */
/*                      mit dem Programm BILD erstellt wur-           */
/*                      den, in MS-Word-Dateien.                      */
/*--------------------------------------------------------------------*/
/*     Autoren        : GÜNTER JÜRGENSMEIER                           */
/*                      MICHAEL TISCHER                               */
/*     Version        : 2.10                                          */
/*     entwickelt am  : 14.03.1986                                    */
/*     letztes Update : 12.02.1989                                    */
/*--------------------------------------------------------------------*/
/*     Erstellung     : cl -c bw.c                                    */
/*                      link /NOE bw+%LIB%\setargv;                   */
/*--------------------------------------------------------------------*/
/*     (Copyright)    : 1989 by GÜNTER JÜRGENSMEIER                   */
/*                      and MICHAEL TISCHER                           */
/**********************************************************************/

/*-- Include-Dateien einbinden ----------------------------------------*/

#include "word.h"     /* Strukturdefinitionen für Word-Datei */
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <malloc.h>
#include <memory.h>
#include <string.h>

/*-- Konstanten ------------------------------------------------------*/

#define FALSE   0
#define TRUE    1                    /* der Wahrheit auf der Spur */

#define TAB       9                  /* Tabulator                 */
#define LF       10
#define LBREAK   11                  /* Zeilenumbruch             */
#define CR       13                  /* Absatzende                */
#define SHYPHEN  31                  /* weicher Trennstrich       */
#define HSPACE  255                  /* festes Leerzeichen        */

#define PSCREEN 87             /* Variantennr. Hardcopy-Absatz */
#define CSCREEN 20            /* Basisnummer der Zeichenvariante */

/*- Konstanten zur Berechnung der Zeichen-Variantennr.  -----*/
#define NORMAL       0
#define INTENSE      1                   /* intensiv wird zu Fett */
#define UNDERL       2  /* unterstrichen bleibt unterstrichen */
#define INVERS       4                   /* invers wird zu kursiv */
#define NOTSTYLED 255       /* noch kein Style-Code ermittelt */

#define BILD_FILE   ".BLD"   /* Dateierweiterung BILD-Dateien */
#define WORD_FILE   "TXT"    /* Dateierweiterung Word-Dateien */
#define DFV         "SCREEN"  /* Name der Druckformatvorlage */

/*-- Makros ----------------------------------------------------------*/

#define ELVEK(x) ( sizeof(x) / sizeof(x[0]) )

/*-- Strukturen ------------------------------------------------------*/

struct ATTRIBUT              /* beschreibt ein Zeichen-Attribut */
{                            /* aus dem Video-RAM               */
 BYTE vg : 3;                           /* Vordergrund-Farbe */
 BYTE in : 1;                      /* Vordergrund-Intensität */
 BYTE bg : 3;                             /* Hintergrundfarbe */
 BYTE bl : 1;                                /* Blinken-Bit */
};

struct SCREEN           /* beschreibt eine Bildschirmposition */
{                       /* im Video-RAM                       */
 BYTE ch;                          /* ASCII-Code des Zeichens */
 struct ATTRIBUT at;                             /* Attribut */
};
```

Listing 2: `BW.C`

```
struct KZP      /* beschreibt einen Kommandozeilen-Parameter */
{
 char *befehl,                  /* Name des Parameters ohne '/' */
 *shortcut;                     /* Abkürzung ebenfalls ohne '/'*/
 BYTE *flag,                    /* Adresse des zugehörigen Flags */
 wert;                                       /* Wert für das Flag */
};

struct RAHMEN                       /* enthält die Rahmenzeichen */
{
 char chVert,                              /* vertikaler Strich */
      chHoriz,                           /* horizontaler Strich */
      chUL,                                /* obere linke Ecke */
      chUR,                               /* obere rechte Ecke */
      chLL,                               /* untere linke Ecke */
      chLR;                              /* untere rechte Ecke */
};

/*-- globale Variablen ---------------------------------------*/

BYTE rahmen  = 1,          /* Rahmen um das Hardcopy ziehen? */
     docujet = FALSE;                   /* nicht für Docujet? */

struct SCREEN scr[25][80];          /* nimmt Hardcopy-Datei auf */
struct RAHMEN *rptr;          /* Pointer auf die Rahmenzeichen */

int handle;                         /* Handle für Dateizugriff */
ULONG pos;              /* Position innerhalb des Ausgabe-Datei */
BYTE  oldstyle;               /* Attribut-Code letztes Zeichen */

int  anzpage;    /* Anzahl der allokierten Pages für FPROPCs */
void *pptr = (void *) 0;     /* Pointer auf allokierte Pages */

/*-- Funktionen ----------------------------------------------*/

/************************************************************
*  Funktion           : W r i t e P a g e                  *
**----------------------------------------------------------**
*  Aufgabe            : Schreibt eine 128-Byte-Page in die  *
*                       Ausgabe-Datei                        *
*  Eingabe-Parameter: PAGE  = Pointer auf die Page          *
*  Return-Wert        : keiner                               *
*  Info               : - Das Handle der Ausgabedatei muß in der*
*                         globalen Variablen HANDLE verzeichnet *
*                         sein.                              *
*                       - Tritt bei der Ausgabe ein Fehler auf, *
*                         wird die Programmausführung abgebr.   *
************************************************************/

void WritePage( BYTE * page )
{          /* die Page in die Ausgabedatei schreiben */
 if ((write(handle, page, PAGELEN)) != PAGELEN)
  {        /* die Page konnte nicht geschrieben werden */
  printf(" Achtung! Fehler bei Schreiben in Ausgabedatei.\n");
  exit( 1 );       /* das Programm mit Fehlercode beenden */
  }
 pos += PAGELEN;    /* alles o.k. Positionsz. inkrementieren */
}

/************************************************************
*  Funktion           : W r i t e V o r s p a n n          *
**----------------------------------------------------------**
*  Aufgabe            : Gibt den Vorspann der Datei nach dem *
*                       Abschluß der Zeichenkonvertierung aus. *
*  Eingabe-Parameter: CPOS  = Offsetadresse des letzten      *
*                             Text-Zeichens innerhalb der    *
*                             Datei.                         *
*                     PAPOS = Offsetadresse innerhalb der    *
*                             Datei, an der die Absatzfor-   *
*                             mate beginnen.                 *
*  Return-Wert        : keiner                               *
*  Info               : - Tritt bei der Ausgabe ein Fehler auf, *
*                         wird die Programmausführung abgebr.   *
************************************************************/

void WriteVorspann( ULONG cpos, ULONG papos )
{
 VORSPANN vorspann;          /* Struktur bildet Page #0 nach */
```

Listing 2: *(Fortsetzung)*

```
memset( &vorspann, '\0', sizeof( vorspann ) ); /* alle F. 0 */
vorspann.wIdent = 48689;                  /* Identitätscode */
vorspann.wTool = 43776;           /* noch ein Identitätscode */
vorspann.fcMac = cpos;           /* Anzahl Textzeichen + 128 */
vorspann.pnPara = ((papos +127) >> 7);  /*Page-Nr. Absatz-F.*/
vorspann.pnFntb =                           /* keine Fußnoten */
vorspann.pnSep  =               /* keine Bereichsbeschreiber */
vorspann.pnSetb =                 /* keine Bereichstabelle */
vorspann.pnPgtb =           /* keine Seitenumbruchs-Tabelle */
vorspann.pnSumd =        /* keine Informationen über den Text */
vorspann.pnMac  = ((pos + 127) >> 7);   /* Anzahl Pages ges. */

/*-- Namen der Druckformatvorlage kopieren -----------------*/
memcpy(vorspann.szSsht, DFV, sizeof(DFV));

lseek( handle, 0L, 0 );        /* File-Pointer an den Anfang */
WritePage( (BYTE *) &vorspann );    /* den Vorspann ausgeben */
}

/************************************************************
*  Funktion           : S t o r e F o r m a t              *
**----------------------------------------------------------**
*  Aufgabe            : Speichert das Format einer Gruppe von *
*                       Zeichen.                             *
*  Eingabe-Parameter: FPTR  = Zeiger auf einen FPROPC       *
*                     LEN   = Anzahl der Bytes im FPROPC    *
*                             exklusive Feld cch             *
*                     POS   = Position des Zeichens hinter   *
*                             dem letzten Zeichen der Gruppe *
*  Return-Wert        : keiner                               *
************************************************************/

void StoreFormat( FPROPC *fptr, int len, ULONG pos )
{
 static FPAGE *ap;               /* Ptr auf die aktuelle Page */
 static int sum;            /* Anzahl der Bytes in der Page */
 static BYTE *ptr;        /* zeigt auf Position für FPROPC */
 ULONG last;      /* letztes Zeichen in einer beendeten Page */
 register int i;                           /* Schleifenzähler */
 BYTE *lptr;          /* Laufzeiger auf die einzelnen FROPCs */

 if ( pptr == (void *) 0 )      /* noch kein Puffer allokiert? */
  {                                                  /* Nein */
  pptr = malloc( PAGELEN );          /* eine Page allokieren */
  if ( pptr == (void *) 0 )                /* kein Speicher? */
   {
    printf("\nAchtung! Zu wenig Speicher!\n");
    exit( 1 );             /* Programm mit Fehlercode beenden */
   }
  else                          /* Page konnte allokiert werden */
   {
    anzpage = 1;                   /* bisher eine Page allokiert */
    ap = (FPAGE *) pptr;                      /* aktuelle Seite */
    sum = sizeof(ap->cfod) + sizeof(ap->fcFirst);
    ap->cfod = 0;               /* noch keinen FOD in der Page */
    ptr = &(ap->cfod);                   /* Position für FPROPC */
    ap->fcFirst = 128;    /* Page beginnt mit ersten Zeichen */
   }
  }

 fptr->cch = len;                /* Länge des FPROPCs speichern */

 /*- paßt noch ein FOD und zugehöriger FPROPC in die Page? --*/
 if (sum + len + 1 + sizeof(FOD) > PAGELEN)
  {                           /* Nein, Page-Buffer vergrößern */
  last = ap->rgfod[ap->cfod-1].fcLim;   /* letztes Zeichen */
  pptr = realloc( pptr, (anzpage+1) * PAGELEN );
  if ( pptr == (void *) 0 )                /* genug Speicher? */
   {                                                 /* Nein */
    printf("\nAchtung! Zu wenig Speicher!\n");
    exit( 1 );             /* Programm mit Fehlercode beenden */
   }
  else
   {
    /*-- nächste Page initialisieren ----------------------*/
    ap = (FPAGE *) pptr+anzpage;            /* aktuelle Seite */
    ++anzpage;          /* Anzahl der Pages inkrementieren */
    sum = sizeof(ap->cfod) + sizeof(ap->fcFirst);
```

Listing 2: *(Fortsetzung)*

```
    ap->fcFirst = last;              /* erstes Zeichen setzen */
    ap->cfod = Ø;              /* noch keinen FOD in der Page */
    ptr = &(ap->cfod);               /* Position für FPROPC */
   }
  }

 /*-- FOD in die Page eintragen ----------------------------*/
 sum += sizeof(FOD);   /* Länge des FODs auf Summme addieren */
 ap->rgfod[ap->cfod].fcLim = pos;            /* letztes Zeichen */
 if ( len == Ø )               /* Standard-Zeichenformat? */
  ap->rgfod[ap->cfod].bfprop = ØxFFFF;                 /* Ja */
 else                                                  /* Nein */
  {
   /*-- testen, ob bereits ein identischer FPROPC in der ----*/
   /*-- Page gespeichert wurde                           ----*/

   for (i=Ø; i<ap->cfod; ++i)  /* die einzelnen FODs durchl. */
    {        /* Pointer auf FPROPC erstellen und vergleichen */
     lptr = (BYTE *) &(ap->rgfod[Ø]) + ap->rgfod[i].bfprop;
     if ( memcmp( fptr, lptr, len+1 ) == Ø )
      break;                 /* einen identischen FPROPC gefunden */
    }
   if ( i < ap->cfod )         /* identischen FPROPC gefunden? */
    /* Ja, Offset zu diesem FPROPC speichern                  */
    ap->rgfod[ap->cfod].bfprop = ap->rgfod[i].bfprop;
   else                                                /* Nein */
    {
     /*-- Position für FPROPC berechnen, Entfernung zu ------*/
     /*-- rgfod[Ø] berechnen und schließlich FPROPC    ------*/
     /*-- kopieren.                                    ------*/
     ptr -= ( len + 1 );                 /* Position berechnen */
     ap->rgfod[ap->cfod].bfprop = ptr-(BYTE *) &(ap->rgfod[Ø]);
     sum += len + 1;     /* Länge auf Gesamtlänge aufaddieren */
     memcpy(ptr, fptr, len+1);     /* FPROPC in Page kopieren */
    }
  }
 ++(ap->cfod); /* Anzahl der FODs in der Page inkrementieren */
}
/***********************************************************
* Funktion          : G e t S t y l e C o d e              *
**---------------------------------------------------------**
* Aufgabe           : Berechnet den Style-Code für ein     *
*                     Zeichen.                             *
* Eingabe-Parameter: ATR = Attribut des Zeichens aus dem   *
*                          Video-RAM                       *
* Return-Wert       : Stylecode                            *
***********************************************************/

BYTE GetStyleCode( struct ATTRIBUT atr )
{
 BYTE style = NORMAL;          /* der zu berechnende Style-Code */

 if (atr.in)                               /* intensive Farbe? */
  style |= INTENSE;                                     /* Ja */
 if (atr.vg==1 && atr.bg==Ø)          /* Zeichen unterstrichen? */
  style |= UNDERL;                                      /* JA */
 else           /* das Zeichen ist nicht unterstrichen */
  if (atr.bg > atr.vg)                /* ist das Zeichen invers? */
   style |= INVERS;                                     /* Ja */

 if ( oldstyle == NOTSTYLED )  /* noch kein Style-Code ber.? */
  oldstyle = style;               /* Nein, diesen Code merken */
 return style;                   /* Style-Code zurückliefern */
}
/***********************************************************
* Funktion          : W r i t e C h A t                    *
**---------------------------------------------------------**
* Aufgabe           : Schreibt ein Zeichen in die Ausgabe-  *
*                     Datei und legt die zugehörigen Attribut-*
*                     Informationen an.                    *
* Eingabe-Parameter: ZEICHEN = Code des Zeichens           *
*                    ATR     = Attribut aus Video-RAM      *
* Return-Wert       : keiner                               *
* Info              : - Tritt bei der Ausgabe ein Fehler auf, *
*                       wird die Programmausführung abgebr.  *
***********************************************************/
```

Listing 2: *(Fortsetzung)*

```
void WriteChAt( BYTE zeichen, struct ATTRIBUT atr )
{
 BYTE len,                                /* Länge des FPROP */
      style;                /* Nummer der Zeichen-Variante */
 FPROPC fpropc;               /* Struktur mit Zeichen-Format */

 if ((write(handle, &zeichen, 1)) != 1)      /* Code schreiben */
  {          /* das Zeichen konnte nicht geschrieben werden */
   printf(" Achtung! Fehler bei Schreiben in Ausgabedatei.\n");
   exit( 1 );             /* das Programm mit Fehlercode beenden */
  }
 ++pos;             /* alles o.k. Positionszähler inkrementieren */

 /*-- Ist das Attribut des Zeichens mit dem des vorherigen --*/
 /*-- identisch, müssen keine Attribut-Informationen ange- --*/
 /*-- legt werden.                                          --*/

 style = GetStyleCode( atr );            /* Style-Code berechnen */

 if ( style == oldstyle )     /* sind die Attribute identisch? */
  return;                         /* Ja, zurück zum Aufrufer */

 /*-- die Attribute sind nicht identisch, es muß ein neuer --*/
 /*-- FOD angelegt werden.                                  --*/

 if ( oldstyle == NORMAL )          /* Standard-Zeichenformat? */
  StoreFormat( &fpropc, Ø, pos-1 ); /* Zeichen-Format merken */
 else                       /* nicht Standard-Zeichenformat */
  {                              /* Varianten-Nummer merken */
   fpropc.chpbf1.fStyled = TRUE;
   fpropc.chpbf1.stc = oldstyle + CSCREEN;
   StoreFormat( &fpropc, 1, pos-1 ); /*Zeichen-Format merken*/
  }

 oldstyle = style;                           /* Style-Code merken */
}

/***********************************************************
* Funktion          : W r i t e C h a r                    *
**---------------------------------------------------------**
* Aufgabe           : Schreibt ein Zeichen ohne besondere  *
*                     Attribute in die Ausgabedatei        *
* Eingabe-Parameter: ZEICHEN = ASCII-Code des Zeichens     *
* Return-Wert       : keiner                               *
* Info              : - Tritt bei der Ausgabe ein Fehler auf, *
*                       wird die Programmausführung abgebr.  *
***********************************************************/

void WriteChar( char zeichen )
{
 struct ATTRIBUT atr;               /* Attribut für WriteChAt() */

 atr.bg = FALSE;                  /* keine besonderen Attribute */
 atr.vg = FALSE;
 atr.in = FALSE;
 atr.bl = FALSE;
 WriteChAt(zeichen, atr);  /* Zeichen über WriteChAt() ausg. */
}

/***********************************************************
* Funktion          : W r i t e C h a r F o r m a t        *
**---------------------------------------------------------**
* Aufgabe           : Schreibt die Zeichenformate in die   *
*                     Ausgabedatei                         *
* Eingabe-Parameter: keiner                                *
* Return-Wert       : keiner                               *
* Info              : - Tritt bei der Ausgabe ein Fehler auf, *
*                       wird die Programmausführung abgebr.  *
***********************************************************/

void WriteCharFormat( void )
{
 register BYTE * lptr;                          /* Laufzeiger */
 int i;                                  /* Schleifenzähler */
 /*-- die einzelnen Pages im Speicher durchlaufen -----------*/
 for ( lptr = (BYTE *) pptr, i = Ø;
       i <anzpage;
       ++i, lptr += PAGELEN )
```

Listing 2: *(Fortsetzung)*

```
 WritePage( lptr );                       /* die Page ausgeben */

 free( pptr );                     /* Speicher wieder freigeben */
 pptr = (void *) 0;                   /* kein Puffer allokiert */
}

/**************************************************************
*  Funktion          : W r i t e P a r a F o r m a t          *
**------------------------------------------------------------**
*  Aufgabe           : Schreibt die Absatzformate in die      *
*                      Ausgabedatei.                          *
*  Eingabe-Parameter: CPOS    = Offsetadresse des letzten     *
*                               Text-Zeichens innerhalb der   *
*                               Datei.                        *
*  Return-Wert       : keiner                                 *
**************************************************************/

void WriteParaFormat( ULONG cpos )
{
 FPAGE page;             /* eine Seite mit Formatinformationen */
 FPROPP *fptr = (FPROPP *) ((BYTE *) &page+125);

 page.cfod = 2;   /* 2 Absätze befinden sich in dieser Datei */
 page.fcFirst = 128;         /* der Text beginnt mit Absatz #1 */
 page.rgfod[0].fcLim = cpos; /* und endet mit dem letzten Z. */
 page.rgfod[0].bfprop = 121;        /* Entfernung zum FPROPP */
 page.rgfod[1].fcLim = cpos+1;   /* der zweite Paragraph be- */
                                 /* steht nur aus der Endem.  */
 page.rgfod[1].bfprop = 0xFFFF;            /* Standardabsatz */

 /*-- FPROPP für Absatz #1 anlegen -------------------------*/
 fptr->cch = 1;                       /* nur ein Byte im FPROP */
 fptr->papbf1.fStyled = 1;   /* Absatz mit Muster formatiert */
 fptr->papbf1.stc = PSCREEN;          /* Nummer der Variante */

 WritePage( (BYTE *) &page );             /* die Page ausgeben */
}


/**************************************************************
*  Funktion          : P r o c e s s F i l e                  *
**------------------------------------------------------------**
*  Aufgabe           : Eine Datei konvertieren                *
*  Eingabe-Parameter: NAME  = Pointer auf den Dateinamen      *
*  Return-Wert       : TRUE, wenn die Datei konvertiert werden *
*                      konnte, sonst FALSE                    *
*  Info              : Der übergebene Dateiname muß mit der   *
*                      Erweiterung BILD_FILE versehen sein.   *
**************************************************************/

BYTE ProcessFile( char * name )
{
 #define SCL (sizeof(scr))     /* Größe des Bildschirmpuffers */

 register int i, j,                       /* Schleifenzähler */
              row,              /* bearbeitete Bildschirmzeile */
              col;                       /* Bildschirmspalte */
 BYTE zeichen;                      /* das jeweilige Zeichen */
 struct ATTRIBUT atr;               /* Attribut des Zeichens */
 FPROPC fpropc;                              /* Dummy-FPROPC */
 ULONG cpos,                   /* Offsetpos. Zeichen-Formate */
       papos;              /* Offsetpos. Paragraphen-Formate */

 /*-- Eingabe-Datei öffnen ---------------------------------*/
 strupr( name );         /* String in Großbuchstaben umwandeln */
 printf( "%s", name );                /* Dateinamen ausgeben */
 if ( (handle = open(name, O_BINARY | O_RDONLY)) < 0 )
  {               /* BLD-Datei konnte nicht geöffnet werden */
   printf(" Achtung! Datei konnte nicht geöffnet werden.\n");
   return FALSE;                       /* mit Fehlercode zurück */
  }

 /*-- Datei konnte geöffnet werden, jetzt laden -------------*/
 if ( read(handle, (void *) scr, SCL) != SCL )
  {
   printf(" Achtung! Ungültige BLD-Datei.\n");
   return FALSE;                       /* mit Fehlercode zurück */
  }
```

Listing 2: *(Fortsetzung)*

```
/*-- zu erstellende Datei öffnen ---------------------------*/
strcpy( strrchr( name, '.' )+1, WORD_FILE );
if ((handle = open(name, O_CREAT | O_BINARY |
      O_TRUNC | O_WRONLY, S_IREAD | S_IWRITE)) < 0)
 {
  printf(" Achtung! Ausgabedatei kann nicht erstellt"
         "werden.\n");
  return FALSE;                       /* mit Fehlercode zurück */
 }

printf(" ---> %s", name );  /* Namen der Ausgabedatei ausg. */
pos = 0;                           /* im ersten Byte der Datei */
oldstyle = NOTSTYLED;          /* bisher kein Style festgelegt */
WritePage( (BYTE *) 0 );            /* Dummy-Vorspann schreiben */

if ( rahmen )             /* Rahmen um das Hardcopy ziehen? */
 {                                                     /* Ja */
  /*-- obere horizontale Zeile ausgeben -------------------*/
  WriteChar( rptr->chUL );                /* obere linke Ecke */
  for (i = 0; i < 82; i++)             /* horizontale Striche */
   WriteChar( rptr->chHoriz );
  WriteChar( rptr->chUR );               /* obere rechte Ecke */
  WriteChar( LBREAK );                    /* Zeilenumbruch */

  /*-- eine weitere Zeile ausgeben -------------------------*/
  WriteChar( rptr->chVert );      /* Zeichen für linken Rand */
  for (i = 0; i < 82; i++)          /* diese Zeile ist leer */
   WriteChar(' ');
  WriteChar( rptr->chVert );     /* Zeichen für rechten Rand */
  WriteChar( LBREAK );                    /* Zeilenumbruch */
 }


/*-- die 25 Zeilen des Hardcopies durchlaufen -------------*/
for (row = 0; row < 25; row++)
 {                                    /* ein Zeile bearbeiten */
  if ( rahmen )                           /* Rahmen ausgeben? */
   {                                                   /* Ja */
    WriteChar( rptr->chVert );   /* Zeichen für linken Rand */
    WriteChar( ' ' );                 /* weiches Leerzeichen */
   }
  else                                       /* kein Rahmen */
   WriteChar( HSPACE );                /* hartes Leerzeichen */

   /*-- die Spalten der Zeile durchlaufen -----------------*/
  for (col = 0; col < 80; col++)
   {                                /* eine Spalte bearbeiten */
    atr = scr[row][col].at;                 /* Attribut laden */
    zeichen = scr[row][col].ch;              /* Zeichen laden */

    /*-- ASCII-Code des Zeichens feststellen und umwandeln -*/
    switch ( zeichen )
     {
      case '\0'    : zeichen = ' ';               /* NUL-Code */
                     break;
      case TAB     : zeichen = 'o';               /* Tabulator */
                     break;
      case SHYPHEN : zeichen = 232;             /* w. Bindestr. */
                     break;
     }


     /*-- bei der Konvertierung für die Arbeit mit DocuJet -*/
     /*-- muß das reverse Leerzeichen umgesetzt werden     -*/

    if ( docujet && atr.bg && zeichen == ' ')
     zeichen = 0x7f;                       /* Bedingung erfüllt */

    WriteChAt( zeichen, atr );              /* Zeichen ausgeben */
   }

  if ( rahmen )                 /* Rahmen um das Hardcopy? */
   {                                                   /* Ja */
    WriteChar( ' ' );              /* Leerzeichen ausgeben */
    WriteChar( rptr->chVert );      /* rechten Rand ausgeben */
    WriteChar( LBREAK );         /* Zeilenumbruch ausgeben */
   }
  else                                       /* kein Rahmen */
```

Listing 2: *(Fortsetzung)*

```
    {
     WriteChar( HSPACE );                 /* hartes Leerzeichen */
     if ( row < 24 )                    /* in der letzten Zeile? */
      WriteChar( LBREAK );   /* Nein, Zeilenumbruch ausgeben */
    }
  }

 if ( rahmen )              /* Rahmen um das Hardcopy ziehen? */
  {                                                    /* Ja */
   /*-- eine weitere Zeile ausgeben --------------------------*/
    WriteChar( rptr->chVert );    /* Zeichen für linken Rand */
    for (i = 0; i < 82; i++)         /* diese Zeile ist leer */
     WriteChar( ' ' );
    WriteChar( rptr->chVert );   /* Zeichen für rechten Rand */
    WriteChar( LBREAK ); /* Zeilenumbruch

   /*-- untere horizontale Zeile ausgeben ------------------*/
   WriteChar( rptr->chLL );               /* untere linke Ecke */
   for (i = 0; i < 82; i++)           /* horizontale Striche */
    WriteChar( rptr->chHoriz );
   WriteChar( rptr->chLR );               /* untere rechte Ecke */
  }

 WriteChar( CR );                        /* Absatzende ausgeben */
 WriteChar( LF );

 cpos = pos;            /* Offsetposition des letzten Zeichens */
                        /* innerhalb der Datei merken          */

 StoreFormat( &fpropc, 0, pos );      /* Zeichen-Format merken */

 /*-- Rest der Page mit Nullen füllen -----------------------*/

 for (i = 0, j = PAGELEN - (pos % PAGELEN); i < j; i++)
  WriteChar( NULL );

 WriteCharFormat();                 /* Zeichenformate schreiben */
 papos = pos;          /* Position der Absatz-Formate merken */
 WriteParaFormat( cpos );            /* Absatz-Formate ausgeben */

 WriteVorspann( cpos, papos );             /* Vorspann ausgeben */
 close( handle );                 /* Ausgabe-Datei schließen */
 printf("\n");                 /* in die nächste Zeile schalten */
 return TRUE;                                     /* alles o.k. */
}

/**********************************************************************
* Funktion          : C o n v e r t                                  *
**------------------------------------------------------------------**
* Aufgabe           : Die angegebenen Dateien konvertieren           *
* Eingabe-Parameter: ARGC  = Anzahl der Argumente                    *
*                    ARGV  = Pointer auf den Vektor mit den          *
*                            Pointern auf die einzelnen Args         *
* Return-Wert       : TRUE, wenn die Dateien konvertiert             *
*                     werden konnten, sonst FALSE                    *
**********************************************************************/

BYTE Convert( unsigned int argc, char **argv )
{
 register unsigned int i;                    /* Schleifenzähler */
 char dateiname[66];         /* Name der zu berabeitenden Datei */
 char *ppos;                  /* Ptr auf den Punkt im Dateinamen */

 for (++argv, i=1; i<argc; ++i, ++argv)
  {                              /* die Argumente durchlaufen */
   if ( **argv != '/' )          /* Kommandozeilen-Parameter? */
    {                            /* Nein, muß Dateiname sein */
     strcpy( dateiname, *argv );          /* Dateiname kopieren */
     if ( strrchr( dateiname, '.' ) == NULL )
      strcat( dateiname, BILD_FILE );             /* Erw. anhängen */
     if ( ProcessFile( dateiname ) == 0)   /* Datei konvert. */
      return FALSE;                                 /* Fehler */
    }
  }
 return TRUE;         /* alle Dateien einwandfrei konvertiert */
}
```

Listing 2: *(Fortsetzung)*

```
/**********************************************************************
* Funktion          : P r o c e s s A r g s                          *
**------------------------------------------------------------------**
* Aufgabe           : Argumente aus der Befehlszeile aus-            *
*                     werten.                                        *
* Eingabe-Parameter: ARGC  = Anzahl der Argumente                    *
*                    ARGV  = Pointer auf den Vektor mit den          *
*                            Pointern auf die einzelnen Args         *
* Return-Wert       : TRUE, wenn die Argumente o.k. sind,            *
*                     sonst FALSE                                    *
* Info              : den Eingaben des Aufrufers folgend wer-        *
*                     den die globalen Flags (RAHMEN, DOCUJET)*
*                     geladen                                        *
**********************************************************************/

BYTE ProcessArgs( unsigned int argc, char **argv )
{
 static struct KZP kzps[] =            /* akzeptierte Parameter */
  {
   { "KEINRAHMEN", "K",  &rahmen,  0    },
   { "RAHMEN",     "R",  &rahmen,  1    },
   { "DOPPELT",    "2",  &rahmen,  2    },
   { "DOCUJET",    "D",  &docujet, TRUE }
  };
 static struct RAHMEN einfach= {'│', '─', '┌', '┐', '└', '┘'},
                      doppelt= {'║', '═', '╔', '╗', '╚', '╝'};

 register unsigned int i, j;                  /* Schleifenzähler */
 unsigned int datnam;                /* Anzahl der Dateinamen */
 BYTE errcode = 0;                                 /* Fehlercode */

 if ( argc < 2 )                       /* zu wenige Argumente? */
  errcode = 1;                  /* keine Dateiname angegeben */
 else
  {
   datnam = 0;                /* noch keinen Dateinamen entdeckt */
   for (++argv, i=1; i<argc; ++i, ++argv)
    {                              /* die Argumente durchlaufen */
     if ( **argv == '/' )          /* Kommandozeilen-Parameter? */
      {                                                    /* Ja */
       strupr( *argv );        /* Umwandlung in Großbuchstaben */
       for (j=0; j<ELVEK(kzps); ++j)  /* Paramter überprüfen */
        {
         if ( strcmp(kzps[j].shortcut, *argv+1) == 0 ||
              strcmp(kzps[j].befehl, *argv+1) == 0 )
          break;         /* Parameter erkannt, Schleife beenden */
        }
       if ( j == ELVEK( kzps ) )           /* Parameter erkannt? */
        {                                                /* Nein */
         errcode = 2;                 /* ungültiger Parameter */
         break;                  /* aus der Schleife ausbrechen */
        }
       else                     /* der Parameter wurde erkannt */
        *kzps[j].flag = kzps[j].wert;              /* Flag setzen */
      }
     else                        /* kein Kommandozeilen-Parameter */
      ++datnam;                         /* muß ein Dateiname sein */
    }
   if ( datnam == 0 )              /* keine Dateinamen gefunden? */
    errcode = 1;                                          /* Nein */
  }


printf("╔═══════════════════════════════════════════"
       "═══════════════╗\n");
printf("║ BW 2.10: BLD-Dateien in Word-Datei"
       "en umwandeln. ║\n");
printf("║ Umrahmung des Hardcopy: %s                  ║\n",
       ( rahmen ) ? "Ja  " : "Nein" );
printf("║ Ausgabe für DocuJet   : %s                  ║\n",
       ( docujet ) ? "Ja  " : "Nein" );
printf("║ (C) 1986-89  G.Jürgensmeier, M.Tischer."
       "         ║\n");
printf("║                           "
       "Alle Rechte vorbehalten. ║\n");
printf("╚═══════════════════════════════════════════"
       "═══════════════╝\n");
```

Listing 2: *(Fortsetzung)*

```
if ( errcode == 1 )             /* keine Dateinamen gefunden? */
{                                                     /* Ja */
  printf("Achtung! Keine Dateinamen angegeben.\n");
  return FALSE;                      /* mit Fehlercode zurück */
}
else
 if ( errcode == 2 )                 /* ungültiger Parameter? */
  {                                                   /* Ja */
   printf("Achtung! Parameter nicht erkannt \"%s\".\n",*argv);
   return FALSE;                     /* mit Fehlercode zurück */
  }
if ( rahmen == 1 )                       /* einfacher Rahmen? */
 rptr = &einfach;    /* Ja, Pointer auf Rahmenzeichen merken */
else                                 /* kein einfacher Rahmen */
 if ( rahmen == 2 )                      /* doppelter Rahmen? */
   rptr = &doppelt; /* Ja, Pointer auf Rahmenzeichen merken */
return TRUE;                                    /* alles o.k. */
}
```

Listing 2: *(Fortsetzung)*

```
/*************************************************************
*                                                            *
*                  H A U P T P R O G R A M M                 *
*                                                            *
*************************************************************/

void main( unsigned int argc, char *argv[] )
{

 if ( ProcessArgs( argc, argv ) )      /* Argumente auswerten */

  if ( Convert( argc, argv ) )       /* Dateien konvertieren */

   exit( 0 );                                   /* alles o.k. */

 exit( 1 );                /* Programm mit Fehlercode verlassen */

}
```

Listing 2: *(Ende)*

Eine komfortable Benutzeroberfläche in C (Teil 2):

Zugriff auf Tastatur und Maus

Nachdem sich der erste Teil dieser Serie mit dem Zugriff auf den Bildschirm beschäftigt hat, ist diese Folge den Eingabegeräten Tastatur und Maus gewidmet. Wiederum möchten wir Ihnen eine Reihe von C-Funktionen vorstellen, die Ihnen bei der Erstellung SAA- bzw. DOS 4.0-konsistenter Benutzeroberflächen wertvolle Dienste leisten.

Windows und MS-DOS 4.0 haben vorgemacht, was schon bald nicht mehr nur als interessantes Feature begrüßt, sondern als fester Bestandteil eines Programms erwartet werden wird: die Unterstützung der Maus als vollwertiges Eingabegerät. Nicht eine einfache Tastatur-Emulation, wie etwa die Umsetzung der Mausbewegung in die Betätigung der entsprechenden Cursortaste, sondern die Beachtung der individuellen Philosophie, die diesem Eingabegerät zugrunde liegt, ist hier gefordert.

Unterschiede zwischen Tastatur und Maus

Bevor man sich an die Entwicklung eines Maus-Interfaces macht, sollte man sich den grundlegenden Unterschied zwischen den Eingabegeräten Maus und Tastatur vor Augen führen. Zwar stellen beide ein Medium dar, mit dessen Hilfe der Anwender Informationen an ein Programm übermitteln kann, doch tragen diese Informationen unterschiedlichen Charakter. Im Fall der Tastatur ist die übermittelte Information der Code einer betätigten Taste, der vom Programm im Rahmen des aktuellen Kontexts interpretiert wird und zu dessen Verarbeitung es keiner weiteren Information bedarf. Anders verhält es sich jedoch mit der Maus. Eine Information, wie z.B. die Nachricht, daß der linke oder rechte Mausknopf niedergedrückt wurde, ist zunächst einmal sinnleer und erhält erst durch die Kombination mit einer anderen Information Gewicht. Indem nämlich die Bildschirmposition und damit das Objekt, über dem sich der Maus-Cursor befindet, in Bezug zu der Maus-Information gesetzt wird, erfährt das Programm, was ihm der Anwender mitteilen möchte.

Wie man diese Philosophie nutzen kann, um einen Großteil der Programmfunktionen, die nicht mit der Eingabe alphanumerischer Informationen zusammenhängen, über die Maus abwickeln zu können, machen grafische Benutzeroberflächen wie etwa MS-Windows vor. Sie stellen auf dem Bildschirm zahlreiche Symbole (Icons) dar, die jeweils mit einer bestimmten Programmfunktion verbunden sind und vom Anwender mit Hilfe des Mauszeigers erreicht und aktiviert werden können.

Zwar können textorientierte Applikationen - und um die geht es in dieser Serie - von dieser Möglichkeit nicht in dem Umfang Gebrauch machen, wie es im Grafikmodus möglich ist, doch bestimmt auch hier die jeweilige Mausposition, wie

```
/*
[ ]--------------------------------------------------------[ ]
 |                                                          |
 |      Include-Datei     : KBM.H                           |
 |                        : zur Einbindung der Funktionen   |
 |                          zum Zugriff auf Tastatur & Maus |
 |      erstellt am       : 25.01.1989                      |
 |      letztes Update am: 30.01.1989                       |
 |                                                          |
 |      (Copyright)       : 1989 by MICHAEL TISCHER         |
 |                                                          |
[ ]--------------------------------------------------------[ ]
*/

/*-- Typedefs ----------------------------------------------*/

#ifndef BYTE                         /* BYTE bereits definiert? */
  typedef unsigned char BYTE;               /* Nein, definieren */
#endif

typedef void (* FKTPTR)( void );   /* Ptr auf VOID-Funktion */

typedef unsigned int TASTE;                      /* Tastencode */

typedef unsigned long PTRVIEW;       /* Maske für Maus-Cursor */

typedef struct {
                 BYTE x1,     /* Koordinaten der oberen linken */
                      y1,     /* und unteren rechten Ecke des  */
                      x2,     /* spezifizierten Bereichs       */
                      y2;
                 long ptr_mask;        /* Maske für Maus-Cursor */
               } BEREICH;

/*-- Konstanten --------------------------------------------*/

#ifndef TRUE             /* TRUE und FALSE bereits definiert? */
  #define TRUE 1                                     /* Nein */
  #define FALSE 0
#endif

/*-- Event-Codes -------------------------------------------*/

#define EV_MOU_MOVE       1                    /* Maus bewegt */
#define EV_LEFT_PRESS     2  /* linker Mausknopf niedergedr. */
#define EV_LEFT_REL       4  /* linker Mausknopf losgelassen */
#define EV_RIGHT_PRESS    8 /* rechter Mausknopf niedergedr. */
#define EV_RIGHT_REL     16 /* rechter Mausknopf losgelassen */
#define EV_MOU_ALL       31             /* alle Maus-Events */
#define EV_KEY_AVAIL     32             /* Taste verfügbar */

#define KEIN_BEREICH 255 /* Maus-Cursor nicht in Bereich xy */

/*-- Codes einiger Tasten, wie sie KbdGetKey() liefert -----*/

#define BEL         7                  /*  Klingelzeichen  */
#define BS          8                  /*  Backspace-Taste */
#define TAB         9                  /*  Tabulator-Taste */
#define LF         10                  /*  Linefeed        */
#define CR         13                  /*  Return-Taste    */
#define ESC        27                  /*  Escape-Taste    */
#define SPACE      32                  /*  Leer-Taste      */
#define CTRL_A      1                  /*  CTRL + A        */
#define CTRL_B      2                  /*  CTRL + B        */
#define CTRL_C      3                  /*  CTRL + C        */
#define CTRL_D      4                  /*  CTRL + D        */
#define CTRL_E      5                  /*  CTRL + E        */
#define CTRL_F      6                  /*  CTRL + F        */
#define CTRL_G      7                  /*  CTRL + G        */
#define CTRL_H      8                  /*  CTRL + H        */
#define CTRL_I      9                  /*  CTRL + I        */
#define CTRL_J     10                  /*  CTRL + J        */
#define CTRL_K     11                  /*  CTRL + K        */
#define CTRL_L     12                  /*  CTRL + L        */
#define CTRL_M     13                  /*  CTRL + M        */
#define CTRL_N     14                  /*  CTRL + N        */
#define CTRL_O     15                  /*  CTRL + O        */
#define CTRL_P     16                  /*  CTRL + P        */
```

Listing 1: KBM.H

```
#define CTRL_Q     17                /* CTRL + Q        */
#define CTRL_R     18                /* CTRL + R        */
#define CTRL_S     19                /* CTRL + S        */
#define CTRL_T     20                /* CTRL + T        */
#define CTRL_U     21                /* CTRL + U        */
#define CTRL_V     22                /* CTRL + V        */
#define CTRL_W     23                /* CTRL + W        */
#define CTRL_X     24                /* CTRL + X        */
#define CTRL_Y     25                /* CTRL + Y        */
#define CTRL_Z     26                /* CTRL + Z        */
#define BACKTAB    271               /* TAB + SHIFT TAB */
#define ALT_Q      272               /* ALT + Q         */
#define ALT_W      273               /* ALT + W         */
#define ALT_E      274               /* ALT + E         */
#define ALT_R      275               /* ALT + R         */
#define ALT_T      276               /* ALT + T         */
#define ALT_Y      277               /* ALT + Y         */
#define ALT_U      278               /* ALT + U         */
#define ALT_I      279               /* ALT + I         */
#define ALT_O      280               /* ALT + O         */
#define ALT_P      281               /* ALT + P         */
#define ALT_A      286               /* ALT + A         */
#define ALT_S      287               /* ALT + S         */
#define ALT_D      288               /* ALT + D         */
#define ALT_F      289               /* ALT + F         */
#define ALT_G      290               /* ALT + G         */
#define ALT_H      291               /* ALT + H         */
#define ALT_J      292               /* ALT + J         */
#define ALT_K      293               /* ALT + K         */
#define ALT_L      294               /* ALT + L         */
#define ALT_Z      300               /* ALT + Z         */
#define ALT_X      301               /* ALT + X         */
#define ALT_C      302               /* ALT + C         */
#define ALT_V      303               /* ALT + V         */
#define ALT_B      304               /* ALT + B         */
#define ALT_N      305               /* ALT + N         */
#define ALT_M      306               /* ALT + M         */
#define F1         315               /* F1-Taste        */
#define F2         316               /* F2-Taste        */
#define F3         317               /* F3-Taste        */
#define F4         318               /* F4-Taste        */
#define F5         319               /* F5-Taste        */
#define F6         320               /* F6-Taste        */
#define F7         321               /* F7-Taste        */
#define F8         322               /* F8-Taste        */
#define F9         323               /* F9-Taste        */
#define F10        324               /* F10-Taste       */
#define CDOWN      336               /* Cursor-Down     */
#define CHOME      327               /* Cursor-Home     */
#define CUP        328               /* Cursor-Up       */
#define CPGUP      329               /* Cursor-Page Up  */
#define CLEFT      331               /* Cursor-Left     */
#define CRIGHT     333               /* Cursor-Right    */
#define CEND       335               /* Cursor-Right    */
#define CPGDN      337               /* Cursor-Page Dn  */
#define DELETE     339               /* DELETE-Taste    */
#define SHIFT_F1   340               /* SHIFT + F1      */
#define SHIFT_F2   341               /* SHIFT + F2      */
#define SHIFT_F3   342               /* SHIFT + F3      */
#define SHIFT_F4   343               /* SHIFT + F4      */
#define SHIFT_F5   344               /* SHIFT + F5      */
#define SHIFT_F6   345               /* SHIFT + F6      */
#define SHIFT_F7   346               /* SHIFT + F7      */
#define SHIFT_F8   347               /* SHIFT + F8      */
#define SHIFT_F9   348               /* SHIFT + F9      */
#define SHIFT_F10  349               /* SHIFT + F10     */
#define CTRL_F1    350               /* CTRL + F1       */
#define CTRL_F2    351               /* CTRL + F2       */
#define CTRL_F3    352               /* CTRL + F3       */
#define CTRL_F4    353               /* CTRL + F4       */
#define CTRL_F5    354               /* CTRL + F5       */
#define CTRL_F6    355               /* CTRL + F6       */
#define CTRL_F7    356               /* CTRL + F7       */
#define CTRL_F8    357               /* CTRL + F8       */
#define CTRL_F9    358               /* CTRL + F9       */
#define CTRL_F10   359               /* CTRL + F10      */
#define ALT_F1     360               /* ALT + F1        */
#define ALT_F2     361               /* ALT + F2        */
```

Listing 1: *(Fortsetzung)*

```
#define ALT_F3     362               /* ALT + F3        */
#define ALT_F4     363               /* ALT + F4        */
#define ALT_F5     364               /* ALT + F5        */
#define ALT_F6     365               /* ALT + F6        */
#define ALT_F7     366               /* ALT + F7        */
#define ALT_F8     367               /* ALT + F8        */
#define ALT_F9     368               /* ALT + F9        */
#define ALT_F10    369               /* ALT + F10       */
#define CTRL_LF    371               /* CTRL-Left       */
#define CTRL_RI    372               /* CTRL-Right      */
#define CTRL_PGDN  374               /* CTRL-PgUp       */
#define CTRL_HOME  375               /* CTRL-Home       */
#define ALT_1      376               /* ALT + 1         */
#define ALT_2      377               /* ALT + 2         */
#define ALT_3      378               /* ALT + 3         */
#define ALT_4      379               /* ALT + 4         */
#define ALT_5      380               /* ALT + 5         */
#define ALT_6      381               /* ALT + 6         */
#define ALT_7      382               /* ALT + 7         */
#define ALT_8      383               /* ALT + 8         */
#define ALT_9      384               /* ALT + 9         */
#define ALT_0      385               /* ALT + 0         */
#define CTRL_PGUP  388               /* CTRL-PgUp       */

/*-- Makros ---------------------------------------------*/

#define MouGetCol() (ev_col)       /* liefern Mausposition & */
#define MouGetRow() (ev_row)       /* -bereich im Moment des */
#define MouGetBereich() (ev_ber)   /* Event-Eintritts        */
#define IsAKey(k) ((k) < 256 )          /* ist K Ascii-Taste? */
#define IsXKey(k) ((k) >= 256) /* ist K erweiterter T.Code? */
#define MouAvail() ( mavail ) /* liefert TRUE, wenn Maus v. */
#define KeyAvail() ( key_avail )         /* Taste verfügbar? */
#define KEY(c) ((TASTE) (c))       /* Typumwandlung für CHAR */
#define MouGetAktCol() ( moucol )  /* liefern jeweils aktu- */
#define MouGetAktRow() ( mourow )  /* elle Mausposition      */
#define MouGetAktBer() ( mouber )

 /*-- ist Taste X mit einer Funktion verknüpft? ------------*/
#define IsFkt(x)\
        (*(fkttab + ((x) >> 3)) & (1 << ((x) & 7)))

/*-- ist Taste X ein Makro? --------------------------------*/
#define IsMakro(x)\
        (*(makrotab + ((x) >> 3)) & (1 << ((x) & 7)))

/*-- Liefert die Anzahl Elemente in einem Vektor X ---------*/
#define ELVEK(x) ( sizeof(x) / sizeof(x[0]) )

/*-- Makros zur Erstellung der Bit-Maske, die das Er-   ----*/
/*-- scheinungsbild des Software-Maus-Cursors definiert ----*/

#define MouPtrMask( z, f )\
  ( (( (PTRVIEW) f) >> 8 << 24) +\
    ((( PTRVIEW) z) >> 8 << 16) +\
    (((f) & 255) << 8) + ((z) & 255) )

#define PTRSAMECHAR   ( 0x00ff )
#define PTRDIFCHAR(z) ( (z) << 8 )

#define PTRSAMECOL    ( 0x00ff )
#define PTRINVCOL     ( 0x7777 )
#define PTRSAMECOLB   ( 0x807f )
#define PTRINVCOLB    ( 0xF777 )
#define PTRDIFCOL(f)  ( (f) << 8 )
#define PTRDIFCOLB(f) ( ((f)|0x80) << 8 )

/*-- Funktionsdeklarationen über Makros --------------------*/

#define MouSetMoveAreaAll()\
                   MouSetMoveArea( 0, 0, tcol-1, tline-1 );

#define KbdUngetKey(x) KbdINextKey(x)

/*-- externe Variablen -------------------------------------*/

extern BYTE * fkttab,
            * makrotab,
```

Listing 1: *(Fortsetzung)*

```
              ev_ber,
              ev_col,
              ev_row,
              moucol,
              mourow,
              mouber,
              mavail,
              key_avail;

extern int    tline,
              tcol;

/*-- Funktions-Deklarationen ------------------------------*/

BYTE KbmInit            ( void );
void KbmEnd             ( void );
int  KbmEventWait       ( int wait_event );
void MouSetBereich      ( BYTE anzahl, BEREICH * ptr );
void MouShowMouse       ( void );
void MouHideMouse       ( void );
void MouSetMoveArea     ( BYTE x1, BYTE y1, BYTE x2, BYTE y2 );
void MouSetSpeed        ( int xspeed, int yspeed );
void MouDefinePtr       ( PTRVIEW mask );
void MouSetDefaultPtr   ( PTRVIEW standard );
void MouSetPtr          ( int col, int row );
void MouPushPara        ( void );
void MouPopPara         ( void );
TASTE KbdGetKey         ( void );
void KbdFlushBuf        ( void );
void KbdClearAllMaks    ( void );
void KbdClearAllFkt     ( void );
BYTE KbdSetMakro        ( TASTE key, TASTE *ptr, BYTE anzahl );
void KbdDelMakro        ( TASTE key );
BYTE KbdSetFkt          ( TASTE key, FKTPTR fktptr );
void KbdDelFkt          ( TASTE key );
void KbdLearn           ( TASTE * lbptr, int len );
int  KbdStopLearn       ( void );
void KbdINextKey        ( TASTE key );
```

Listing 1: *(Ende)*

Nr	Aufgabe
00h	Reset des Maustreibers
01h	Maus-Cursor auf dem Bildschirm anzeigen
02h	Maus-Cursor vom Bildschirm entfernen
03h	Mausposition und Status der Mausknöpfe ermitteln
04h	Position des Maus-Cursors festlegen
05h	Information über Betätigung der Mausknöpfe ermitteln
06h	Information über Loslassen der Mausknöpfe ermitteln
07h	horizontale Bewegungsgrenzen für Maus-Cursor festlegen
08h	vertikale Bewegungsgrenzen für Maus-Cursor festlegen
09h	Erscheinungsbild des Maus-Cursors im Grafikm. definieren
0Ah	Erscheinungsbild des Maus-Cursors im Textmodus festlegen
0Bh	relative Mausbewegung ermitteln
0Ch	benutzerdefinierten Event-Handler installieren
0Dh	Emulation des Lichtgriffels anschalten
0Eh	Emulation des Lichtgriffels ausschalten
0Fh	Mausgeschwindigkeit festlegen
10h	Bereich festlegen, in dem der Maus-Cursor nicht erscheint
11h	nicht definiert
12h	nicht definiert
13h	Faktor für Verdoppelung der Maus-Geschwindigkeit setzen
14h	benutzerdefinierte Event-Handler austauschen
15h	Größe des Maus-Status-Puffers ermitteln
16h	Maus-Status in einem Puffer des Aufrufers sichern
17h	Maus-Status aus einem Puffer des Aufrufers restaurieren
18h	weitere Event-Handler installieren
19h	Adresse eines bestimmten Event-Handlers ermitteln
1Ah	Maus-Geschwindigkeit und Faktor für Verdoppelung festlegen
1Bh	aktuelle Maus-Geschwindigkeit ermitteln
1Ch	Interrupt-Rate des Maustreibers festlegen
1Dh	Bildschirmseite für Maus-Cursor definieren
1Eh	Bildschirmseite des Maus-Cursors ermitteln
1Fh	Maustreiber deaktivieren
20h	Maustreiber aktivieren
21h	Reset des Maustreibers ohne Hardware-Initialisierung
22h	Fremdsprache für Treiber-Meldungen festlegen
23h	Fremdsprache für Treiber-Meldungen ermitteln
24h	Information über Maus-Hardware und -Treiber ermitteln

Tabelle 1: *Die Funktionen des Maustreibers*

das Programm beispielsweise auf die Betätigung eines Mausknopfs zu reagieren hat. Darum muß ein Maus-Interface über die Möglichkeit verfügen, den Bildschirm in verschiedene Bereiche (Objekte) zu unterteilen, um einem übergeordneten Modul bei der Betätigung eines Mausknopfs mitteilen zu können, welches Objekt der Anwender ausgewählt hat. Dies aber wirft die Frage auf, wie ein Maus-Interface von der Betätigung eines Mausknopfes Notiz nehmen und die aktuelle Mausposition ermitteln kann.

Kommunikation zwischen Maus-Interface und Maus-Hardware

Als Mittler zwischen einem Programm und der Maus-Hardware fungiert ein Maustreiber, der entweder beim Booten des Systems als Gerätetreiber installiert, oder als eigenständiges (TSR-) Programm von DOS aus gestartet werden kann. Er überwacht ständig die Bewegungen der Maus sowie den Status der Mausknöpfe und bewegt den Maus-Cursor parallel zur Maus über den Bildschirm. Da dieser Treiber direkt auf die Hardware der Maus eingehen muß und sich die Mäuse der verschiedenen Hersteller zum Teil stark unterscheiden, ist ein solcher Treiber in der Regel im Lieferumfang einer Maus enthalten.

Trotz der unübersehbaren Unterschiede im Hinblick auf die Maus-Hardware, unterstützen die Maustreiber fast aller Maus-Hersteller ein Software-Interface, das dem der Microsoft-Maus zumindest im Hinblick auf die unterstützten Funktionen und deren Aufruf nachempfunden ist. Als Schnittstelle zwischen einem Programm und dem Maustreiber dient dabei der Interrupt 33h, über den die verschiedenen Funktionen des Maustreibers aufgerufen werden können.

Wie beim Software-Interface des DOS-Kerns (Interrupt 21h) erfolgt der Informationsaustausch dabei mit Hilfe der verschiedenen Prozessorregister. Dadurch ist es auch einem C-Programm möglich, die einzelnen Maus-Funktionen mit den Bibliotheksfunktionen `int86` und `int86x` aufzurufen.

Wie *Tabelle 1* zeigt, stellt der Maustreiber einem Programm eine Reihe verschiedenster Funktionen zur Verfügung, von denen jedoch nur einige wenige in jedem Fall benötigt werden. Auch kann man nicht davon ausgehen, daß alle der oben genannten Funktionen von jedem Maustreiber unterstützt werden, da die hier vorgestellten Funktionen komplett erst ab der Version 6 des Microsoft-Maustreibers verfügbar sind. Aus diesem Grund beschränkt sich das hier vorgestellte Modul auf die Arbeit mit den Funktionen 00h bis 0Fh, die auch von älteren Versionen des Maustreibers



Bild 1: *Die Markierung eines Bereichs mit der Maus in KBMDEMO.*

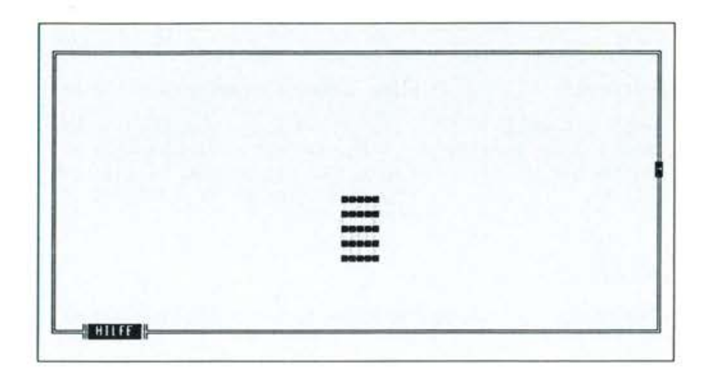


Bild 2: *Es können Bildschirmbereiche definiert werden, in denen sich der Maus-Cursor automatisch verändert.*

unterstützt werden. Falls Sie an den Aufrufkonventionen der einzelnen Funktionen interessiert sind, schlagen Sie bitte in dem Listing auf den folgenden Seiten nach, in dem diese Aufrufe jeweils ausführlich dokumentiert werden.

Das C-Modul KBM.C

Alle Funktionen zur Abfrage der Maus und Tastatur, die im Rahmen dieses Artikels vorgestellt werden, finden sich in dem C-Modul KBM.C, das auf den vorhergehenden und folgenden Seiten abgedruckt ist (*Listings 1 und 2*). `KBM` steht dabei für *K*ey*b*oard & *M*ouse und stellt gleichzeitig das Präfix dar, das allen Funktionen innerhalb des Moduls vorausgeht, die sich auf das Modul als Ganzes bzw. sowohl auf die Tastatur als auch auf die Maus beziehen, wie etwa `KbmInit()` zur Initialisierung des Moduls.

Aus der Sicht eines übergeordneten Moduls stehen die Funktionen `KbmEventWait()` und `GetKey()` im Mittelpunkt des `KBM`-Moduls. Erstere dient dabei zum Warten auf ein bestimmtes Ereignis; auf ein Event, wie man im Englischen sagt. Dieses Ereignis kann entweder die Betätigung einer Taste oder ein Ereignis im Zusammenhang mit der Maus sein, wobei folgende Ereignisse erfaßt werden können:

- die Bewegung der Maus
- die Betätigung des linken oder rechten Mausknopfes
- das Loslassen des linken oder rechten Mausknopfes

Beim Aufruf von `KbmEventWait()` wird das zu erwartende Ereignis in Form einer Bitmaske angegeben, die Sie durch Oder-Verknüpfung der verschiedenen Ereignis-Konstanten aus der Include-Datei `KBM.H` erstellen können. Der Aufruf
`KbmEventWait( EV_KEY_AVAIL | EV_LEFT_PRESS)`
würde `KbmEventWait()` z.B. dazu veranlassen, auf die Betätigung einer Taste **oder** die Betätigung des linken Mausknopfs zu warten. Erst wenn eines dieser Ereignisse eingetreten ist, kehrt die Funktion zum Aufrufer zurück, wobei sie eine Bitmaske übergibt, aus der der Aufrufer die Art des bzw. der eingetretenen Ereignisse ermitteln kann.

Die Art, wie `KbmEventWait()` auf den Eintritt des bzw. der Ereignisse wartet, unterscheidet sich etwas von der Art und Weise, wie man derartige Aufgabenstellungen sonst zu lösen pflegt. Normalerweise würde man die Geräte Maus und Tastatur in einer Schleife so lange direkt abfragen, bis eine der dazu aufgerufenen Funktionen anzeigt, daß das erwartete Ereignis eingetreten ist. Diese Vorgehensweise (man bezeichnet sie als Polling) bringt aber nicht nur bei ihrer Realisation in Hochsprachen den Nachteil mit sich, daß die Abfrageschleife unter Umständen zu langsam ist, um unmittelbar auf das Ereignis reagieren zu können - daß beispielsweise zwar die Betätigung des Mausknopfes bemerkt wird, die Maus sich zu diesem Zeitpunkt aber

```
/****************************************************************/
/*                          K B M . C                           */
/*--------------------------------------------------------------*/
/*    Aufgabe        : Stellt verschiedene Funktionen zur       */
/*                     Abfrage von Maus und Tastatur be-        */
/*                     reit.                                    */
/*--------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                          */
/*    entwickelt am  : 25.01.1989                               */
/*    letztes Update :  2.02.1989                               */
/*--------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] KBM.C /C                */
/*                     dann mit einem anderen Modul linken      */
/****************************************************************/

/*-- Include-Dateien einbinden ---------------------------------*/

#include "vio.h"         /* Zugriff auf die Video-Funktionen */
#include "kbm.h"           /* eigene Include-Datei einbinden */
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <memory.h>

/*-- Typedefs --------------------------------------------------*/

#ifndef WORD                            /* WORD bereits definiert? */
  typedef unsigned int WORD;                   /* Nein, definieren */
#endif

typedef void (interrupt far *INTHANDLER)();

typedef void (_saveregs _loadds far * MOUHAPTR)( void );
```

Listing 2: `KBM.C`

```
typedef struct                /* nimmt aktuelle Mausparameter auf */
 {
  BYTE anz_bereich;                        /* Anzahl der Bereiche */
  BEREICH * akt_bereich;     /* Pointer auf Bereichs-Vektor */
  PTRVIEW stdptr;           /* aktueller Standard-Maus-Cursor */
  BYTE x1,                  /* Eck-Koordinaten der aktuellen */
       y1,                  /* Maus-Area                     */
       x2,
       y2;
 } MOUPARA;

typedef struct   /* beschreibt einen Tasten-Funktionsaufruf */
 {
  TASTE key;                          /* die jeweilige Taste */
  FKTPTR fkt;             /* Pointer auf auzurufende Funktion */
 } KEY_FKT;

typedef struct              /* beschreibt ein Tastaturmakro */
 {
  TASTE key;                      /* die umzusetzende Taste */
  BYTE anzahl;              /* Anzahl der erzeugten Tasten */
  TASTE *bufptr;       /* Pointer auf Vektor mit Makro-Tasten */
 } KEY_MAK;

/*-- Konstanten -------------------------------------------*/

#define MAX_STACK 20              /* Größe des Parameter-Stack */
#define KB_LEN    100           /* Länge des Tastatur-Puffers */
#define KB_INTR   0x09                  /* Tastatur-Interrupt */
#define CTRLB_INT 0x1b                /* Ctrl-Break-Interrupt */
#define CTRLC_INT 0x23                    /* Ctrl-C-Interrupt */
#define KB_INTR   0x09                  /* Tastatur-Interrupt */

#define KEY_EXPANDED ( 32768 )  /* Taste bereits expandiert */

/*-- Makros -----------------------------------------------*/

#define MOUINT(rin, rout) int86(0x33, &rin, &rout)
#define MOUINTX(rin, rout, sr) int86x(0x33, &rin, &rout, &sr)

#define KbEmpty() ( kbend == kbstart )


/*-- Makros zur Umrechnung zwischen Maus-Koordinaten in ----*/
/*-- Bezug auf den virtuellen Maus-Bildschirm und dem   ----*/
/*-- Textbildschirm                                     ----*/

#define XTOCOL(x) ( (x) >> xshift)        /* X durch 2^xshift */
#define YTOROW(y) ( (y) >> 3 )              /* Zeile durch 8 */
#define COLTOX(c) ( (c) << xshift )         /* C mal 2^xshift */
#define ROWTOY(r) ( (r) << 3)                 /* Zeile mal 8 */


/*-- globale Variablen ------------------------------------*/

int tline,                             /* Anzahl Text-Zeilen */
    tcol;                             /* Anzahl Text-Spalten */

BYTE ma_x1,           /* Koordinaten der aktuellen Maus-Area */
     ma_y1,           /* (Bildschirmbereich, in dem sich die */
     ma_x2,           /* Maus bewegen darf)                  */
     ma_y2,
     xshift = 3,  /* Shift-Zähler für Koordinatenumrechnung */
     mavail = FALSE;         /* ist TRUE, wenn Maus verfügbar */

/*-- Maske für den Standard-Maus-Cursor -------------------*/
PTRVIEW stdptr = MouPtrMask( PTRSAMECHAR, PTRINVCOL );

BYTE * bpuf,         /* Ptr auf Puffer für Bereichs-Erkennung */
     anz_bereich = 0;     /* bisher keine Bereiche definiert */
BEREICH * akt_bereich;  /* Pointer auf akt. Bereichs-Vektor */

MOUPARA moustack[ MAX_STACK ];              /* Parameter-Stack */
MOUPARA * stackptr = moustack;  /* Stackpointer für P-Stack */

INTHANDLER OldKbHandler, /* Ptr. auf alten Tast-Int-Handler */
           OldCtrlCHandler,           /* alter Ctrl-C-Handler */
           OldCtrlBHandler;       /* alter Ctrl-Break-Handler */
```

Listing 2: *(Fortsetzung)*

```
BYTE key_avail = FALSE,   /* ist TRUE, wenn Taste verfügbar */
     ctrl_break = FALSE; /* ist TRUE, wenn Ctrl-Break gedr. */
TASTE key_buf[ KB_LEN ];          /* interner Tastaturpuffer */
TASTE * kbstart = key_buf,     /* Pointer auf nächstes und   */
      * kbend   = key_buf;     /* letztes Zeichen im KEY_BUF */

BYTE makroanz = 0,                /* noch kein Makro definiert */
     fktanz = 0,             /* noch keine Funktion definiert */
     *makrotab,          /* Pointer auf Bit-Tabelle für Makros */
     *fkttab;        /* Pointer auf Bit-Tabelle für Funktionen */

KEY_FKT *fktvec;     /* Ptr auf Vektor mit Fkt.-Beschreibern */
KEY_MAK *makvec;    /* Ptr auf Vektor mit Makro-Beschreibern */

TASTE * lernbuf,                  /* Pointer auf Lern-Puffer */
      * lbakt;             /* Pointer auf aktuelle Position */
int lblen;                 /* Länge des Lernbuf in TASTEn ^ */
BYTE lern = FALSE;                         /* Lern-Modus aus */

/*-- Variablen, die mit jedem Aufruf des Maus-Handlers -----*/
/*-- geladen werden                                      -----*/

BYTE mouber = KEIN_BEREICH,          /* aktueller Maus-Bereich */
     moucol,                /* Mausspalte (Text-Bildschirm) */
     mourow;                /* Mauszeile  (Text-Bildschirm) */
int  mouevent;                                 /* Event-Maske */

/*-- Variablen, die nur bei Eintritt eines erwarteten  -----*/
/*-- Events durch den Maus-Handler geladen werden       -----*/

BYTE ev_ber,       /* Bereich, in dem sich die Maus befindet */
     ev_col,                                    /* Mausspalte */
     ev_row;                                     /* Mauszeile */

/***********************************************************
*  Funktion         : K b d H a n d l e r                  *
**---------------------------------------------------------**
*  Aufgabe          : Dies ist der neue Tastatur-Handler,   *
*                     der bei jeder Betätigung einer Taste  *
*                     in Form des Interrupts 0x9 durch die  *
*                     PC-Hardware aufgerufen wird.          *
*  Eingabe-Parameter: keine                                 *
*  Return-Wert      : keiner                                *
*  Info             : - Diese Funktion ist nur zum Aufruf   *
*                       durch die PC-Hardware bestimmt und  *
*                       darf nicht von einer anderen Fkt.   *
*                       aus aufgerufen werden.              *
***********************************************************/

#pragma check_stack(off)         /* kein Stack-Checking hier */

void interrupt far KbdHandler(es, ds, di, si, bp, sp, dx,
                              cx, ax, ip, cs, flags )

WORD es, ds, di, si, bp, sp, dx, cx, ax, ip, cs, flags;
{
 (*OldKbHandler)();      /* alten Interrupt-Handler aufrufen */

 if ( ctrl_break )             /* wurde Ctrl-Break betätigt? */
  {                                                     /* Ja */
   ctrl_break = FALSE;                  /* Flag zurücksetzen */
   _bios_keybrd( _KEYBRD_READ );          /* Break-Code holen */
  }

 if ( key_avail == FALSE )          /* keine Taste verfügbar? */
  key_avail = ( _bios_keybrd( _KEYBRD_READY ) != 0 );
}

#pragma check_stack     /* alten Zustand im Hinblick auf das */
#pragma check_stack     /* Stack-Checking wieder herstellen  */

/***********************************************************
*  Funktion         : M o u H a n d l e r                  *
**---------------------------------------------------------**
*  Aufgabe          : Dies ist der Maus-Handler, der durch  *
*                     den installierten Maustreiber bei     *
*                     Eintritt eines bestimmten Ereignisses *
*                     aufgerufen wird.                      *
```

Listing 2: *(Fortsetzung)*

```
*  Eingabe-Parameter: AX-Register = Event-Maske                *
*                     BX-Register = Status der Maus-Knöpfe     *
*                     CX-Register = Maus-Spalte im virt. B.    *
*                     DX-Register = Maus-Zeile im virt. B.     *
*                     SI-Register = vertikale Mausbewegung     *
*                     DI-Register = horizontale Mausbewegung   *
*  Return-Wert      : keiner                                   *
*  Info             : - Diese Funktion ist nur zum Aufruf      *
*                       durch den Maustreiber bestimmt und     *
*                       darf nicht von einer anderen Fkt.      *
*                       aus aufgerufen werden.                 *
*                     - Die Funktion muß über genau eine       *
*                       lokale Variable verfügen, die auf      *
*                       dem Stack ein WORD beansprucht. An-    *
*                       derenfalls müssen die Konstanten       *
*                       zum Zugriff auf die Register ver-      *
*                       ändert werden.                         *
***************************************************************/

#pragma check_stack(off)         /* kein Stack-Checking hier */

#define REG(x) (*((unsigned *) &dummy+x))
#define AX REG(10)  /* Konstanten zum Zugriff auf die Pro-   */
#define BX REG(7)   /* zessorregister, die durch das Attri- */
#define CX REG(9)   /* but _SAVEREGS nach dem Aufruf der     */
#define DX REG(8)   /* Funktion auf dem Stack gesichert      */
#define SI REG(4)   /* werden                                */
#define DI REG(3)

void _saveregs _loadds far MouHandler( void )
{
 unsigned dummy;       /* darf nicht entfernt werden! (s.o.) */

 mouevent = AX;                          /* Event-Maske merken */
 moucol = XTOCOL( CX );  /* Spalte in Textspalten umrechnen */
 mourow = YTOROW( DX );    /* Zeile in Textzeilen umrechnen */

 /*-- Bereich ermitteln, in dem sich die Maus befindet und -*/
 /*-- feststellen, ob sich der Bereich seit dem letzten    -*/
 /*-- Aufruf des Handlers verändert hat. In diesem Fall    -*/
 /*-- wird eine neue Maske für das Erscheinungsbild des    -*/
 /*-- Maus-Cursors gesetzt.                                -*/

 dummy = *(bpuf + mourow * tcol + moucol);    /* Ber. holen */
 if ( dummy != mouber )                    /* neuer Bereich? */
  MouDefinePtr( (dummy == KEIN_BEREICH) ?              /* Ja */
                   stdptr : (akt_bereich+dummy)->ptr_mask );
 mouber = dummy;  /* Bereichsnummer in globaler Var. merken */
}

#pragma check_stack    /* alten Zustand im Hinblick auf das */
#pragma check_stack    /* Stack-Checking wieder herstellen  */


/**************************************************************
*  Funktion         : B r e a k H a n d l e r                  *
**------------------------------------------------------------**
*  Aufgabe          : Wird bei Betätigung von Ctrl-Break       *
*                     und Ctrl-C aufgerufen.                   *
*  Eingabe-Parameter: keine                                    *
*  Return-Wert      : keiner                                   *
*  Info             : - Kehrt unmittelbar zum Aufrufer zu-     *
*                       rück und verhindert dadurch die        *
*                       Programmbeendigung.                    *
***************************************************************/

#pragma check_stack(off)         /* kein Stack-Checking hier */

void interrupt far BreakHandler(es, ds, di, si, bp, sp, dx,
                                cx, ax, ip, cs, flags )

WORD es, ds, di, si, bp, sp, dx, cx, ax, ip, cs, flags;
{
 ctrl_break = TRUE;              /* Ctrl-Break wurde betätigt */
}

#pragma check_stack    /* alten Zustand im Hinblick auf das */
#pragma check_stack    /* Stack-Checking wieder herstellen  */
```

Listing 2: *(Fortsetzung)*

bereits wieder an einer anderen Bildschirmposition befindet und dadurch auf ein falsches Bildschirmobjekt Bezug genommen wird. Ein anderer Nachteil des Pollings macht sich vor allem in Multitasking-Umgebungen wie z.B. MS-OS/2 sehr unangenehm bemerkbar: die CPU wird fortwährend mit der Ausführung der Abfrageschleife beschäftigt, ohne dabei produktiv zu sein und ihre Arbeitskraft anderen Prozessen zuwenden zu können.

Aus diesem Grund wendet bereits das ROM-BIOS des PC, aber auch der Gerätetreiber der Maus, zur Abfrage des jeweiligen Geräts eine andere Technik an. Indem die Geräte nicht permanent abgefragt, sondern die entsprechende Abfrageroutine von dem Gerät selbst über einen Interrupt immer nur dann aufgerufen wird, wenn ein Ereignis eingetreten ist, wird nicht unnötig Prozessorzeit verschwendet, das Ereignis aber trotzdem unmittelbar nach seinem Eintritt erkannt.

Nicht anders geht das KBM-Modul vor, indem es einen eigenen Interrupt-Handler für die Tastatur und die Maus installiert. Der Tastatur-Handler mit dem Namen `KbdHandler()` wird dabei in die Kette der Interrupt-Handler des Interrupt 09h eingeklingt, der von der Tastatur immer dann aufgerufen wird, wenn ein Taste niedergedrückt oder losgelassen wird. Wie in der letzten Folge des Microsoft System Journals in dem Artikel »Speicherresidente Programme in MS-C« gezeigt wurde, unterstützt der Microsoft C-Compiler ab der Version 5.0 die Entwicklung derartiger Interrupt-Routinen.

Durch das Befehlswort `interrupt` wird es möglich, eine Routine als Interrupt-Routine zu kennzeichnen. Die einzelnen Prozessorregister können dann nicht nur als lokale Variablen angesprochen und manipuliert werden, sondern die Funktion wird auch durch den Maschinensprache-Befehl `iret` beendet, der die Ausführung wieder mit dem Programm fortsetzt, das durch den Aufruf des Interrupts unterbrochen wurde. Zusätzlich generiert MSC Maschinencode, mit dessen Hilfe beim Aufruf der Interrupt-Routine die Segmentadresse des globalen Variablensegments des C-Programms in das DS-Register geladen wird und so auch innerhalb der Interrupt-Routine der Zugriff auf globale Variablen möglich ist.

Installiert wird der Interrupt-Handler `KbdHandler()` innerhalb der Initialisierungs-Funktion des KBM-Moduls, der Funktion `KbmInit()`. Mit Hilfe der Bibliotheksfunktion `_dos_getvect()` wird dort zunächst die Adresse des bisherigen Interrupt-Handlers des Interrupts 09h ermittelt und in der globalen Variablen `OldKbHandler` gespeichert. Der neue Interrupt-Handler wird danach mit Hilfe der Bibliotheksfunktion `_dos_setvect()` installiert.

Innerhalb des neuen Tastatur-Handlers wird jeweils zunächst der alte Interrupt-Handler aufgerufen, dessen Funktionen weder ersetzt werden sollen, noch können. Danach wird anhand der globalen Variablen `key_avail` überprüft, ob kein Zeichen im internen Tastaturpuffer des KBM-Moduls bereitsteht. Ist dies der Fall, wird anhand der

Funktion 01h des BIOS-Tastatur-Interrupts überprüft, ob jetzt ein Zeichen zur Verfügung steht und dementsprechend das Flag `key_avail` auf TRUE oder FALSE gesetzt.

Die Installation dieses Interrupt-Handlers macht es übrigens erforderlich, daß vor der Beendigung eines Programms, das mit den Funktionen aus KBM-Modul arbeitet, die Funktion `KbmEnd( )` aufgerufen wird, in deren Verlauf wieder der alte Interrupt-Handler des Interrupt 09h installiert wird. Bleibt der KBM-Handler über das Ende des Programms hinaus installiert, kommt es spätestens beim Start eines neuen Programms zum Systemabsturz, da dann der Code des Interrupt-Handlers von diesem Programm überschrieben wird.

Nicht ganz so leicht ist es, einen Interrupt-Handler für die Maus zu installieren, da es sich hier nicht im eigentlichen Sinne um eine Interrupt-Routine handelt. Vielmehr wird dieser Handler nicht direkt von der Maus-Hardware, sondern durch den Maustreiber aufgerufen, der speziell zur Installation eines solchen Handlers die Funktion 0Ch bereitstellt.

Der Aufruf des Handlers erfolgt dabei nicht über den Maschinensprache-Befehl `int`, sondern über einen sogenannten FAR-Call, wodurch die Routine auch nicht durch den Befehl `iret`, sondern durch den Befehl `retf` beendet werden muß. Deshalb muß die Funktion als `FAR` deklariert und kann nicht als MSC-Interrupt-Funktion kodiert werden, was weitreichende Folgen hat, wie gleich noch zu sehen ist.

```
                     S T A C K
                  :                       :
                  |-----------------------|
                  | Rücksprungadresse zum |
                  | Aufrufer (FAR)        |
SS:BP+18          |          A X          |
SS:BP+16          |          C X          |
SS:BP+14          |          D X          |
SS:BP+12          |          B X          |
SS:BP+10          |          S P          |
SS:BP+8           |          B P          |
SS:BP+6           |          S I          |
SS:BP+4           |          D I          |
SS:BP+2           |          D S          |
SS:BP      --->   |          E S          |
SS:BP-2    --->   |   lokale Variablen    |
                  :                       :
```

Bild 3: *Stackaufbau beim Aufruf einer C-Funktion, die mit dem Attribut* `_saveregs` *versehen wurde*

```
/************************************************************
*  Funktion          : K b m I B u f F i l l                *
**--------------------------------------------------------**
*  Aufgabe           : Speichert den Bereichscode für einen *
*                      bestimmte Bildschirmbereich.         *
*  Eingabe-Parameter: x1, y1 = Eckkoordinaten des Bild-     *
*                      x2, y2   schirmbereichs              *
*                      CODE   = Bereichscode.               *
*  Return-Wert       : keiner                               *
*  Info              : Diese Funktion darf nur innerhalb    *
*                      dieses Moduls aufgerufen werden.     *
************************************************************/

static void KbmIBufFill( BYTE x1, BYTE y1,
                         BYTE x2, BYTE y2, BYTE code )
{
 register BYTE * lptr;       /* Laufzeiger in den Bereichsp. */
 BYTE i, j;                               /* Schleifenzähler */

 /*-- die einzelnen Zeilen durchlaufen ---------------------*/

 for (j=x2 - x1 + 1 ; y1 <= y2; ++y1)
  {         /* die einzelnen Elemente der Zeile durchlaufen */
   for (lptr = bpuf+y1*tcol+x1, i = j; i ; --i)
    *(lptr++) = code;                    /* den Code setzen */
  }
}

/************************************************************
*  Funktion          : M o u S e t B e r e i c h            *
**--------------------------------------------------------**
*  Aufgabe           : Erlaubt die Definition verschiedener *
*                      Bildschirmbereiche die zur Arbeit    *
*                      mit der Maus mit einem bestimmten    *
*                      Code versehen werden.                *
*  Eingabe-Parameter: - ANZAHL = Anzahl der Bildschirmber.  *
*                      - PTR    = Pointer auf Vektor mit Ele- *
*                                 menten vom Typ BEREICH.   *
*  Return-Wert       : keiner                               *
*  Info              : - Den freibleibenden Bildschirmb. wird *
*                        der Code KEIN_BEREICH zugewiesen.  *
*                      - Beim Eintritt in einen der angege- *
*                        benen Bildschirmbereiche schaltet  *
*                        der Maus-Handler automatisch auf das *
*                        Erscheinungsbild des Maus-Cursors um,*
*                        das innerhalb der einzelnen Kompo- *
*                        nenten des Vektors angegeben wird. *
*                      - Den jeweils aktuellen Bereich können *
*                        Sie nach der Rückkehr aus der Funk- *
*                        tion KbmEventWait() mit Hilfe der  *
*                        Funktion MouGetBereich() ermitteln. *
*                        Der erste Bereich innerhalb des Vek- *
*                        tors trägt dabei die Nummer 0, der *
*                        zweite die Nummer 1 usw.           *
************************************************************/

void MouSetBereich( BYTE anzahl, BEREICH * ptr )
{
 register BYTE i;                        /* Schleifenzähler */

 akt_bereich = ptr;            /* Pointer auf Vektor und An- */
 anz_bereich = anzahl;         /* zahl der Bereiche merken   */
 KbmIBufFill( 0, 0, tcol-1, tline-1, KEIN_BEREICH );
 for (i=0 ; i<anzahl ; ++ptr )
  KbmIBufFill( ptr->x1, ptr->y1, ptr->x2, ptr->y2, i++);
}


/************************************************************
*  Funktion          : K b m E v e n t W a i t              *
**--------------------------------------------------------**
*  Aufgabe           : Wartet auf den Eintritt eines be-    *
*                      stimmten Ereignisses im Zusammenhang *
*                      mit Tastatur und Maus.               *
*  Eingabe-Parameter: WAIT_EVENT = Bit-Maske die das zu er- *
*                                  wartende Ereignis spezi- *
*                                  fiziert.                 *
*  Return-Wert       : Bit-Maske der eingetretenen Ereignisse *
```

Listing 2: *(Fortsetzung)*

```
*  Info               : - WAIT_EVENT kann durch Oder-Verknüpf- *
*                         ung der verschiedenen Konstanten wie *
*                         z.B. EV_MOU_MOVE aus der Datei KBM.H *
*                         gebildet werden                      *
*                       - Die Funktion kehrt erst dann zum     *
*                         Aufrufer zurück, wenn eines oder     *
*                         mehrere der angegebenen Ereignisse   *
*                         eintreten.                           *
***************************************************************/

int KbmEventWait( int wait_event )
{
 int akt_event;                  /* die jeweils aktuelle Event-Maske */
 register BYTE spalte = moucol,         /* letzte Mausposition */
               zeile  = mourow;
 BYTE ende = FALSE; /* wird TRUE, wenn Ereignis eingetreten */

 while ( !ende )   /* wiederholen, bis Ereignis eingetreten */
  {
   /*-- warten bis eines der Ereignisse eintritt -----------*/

   while ( (((akt_event = mouevent) & wait_event) == 0) &&
          !(((wait_event & EV_KEY_AVAIL) !=0) &&
              key_avail!=FALSE) )
    ;

   akt_event &= wait_event;  /* nur Event-Bits stehenlassen */
   if (((wait_event & EV_KEY_AVAIL) !=0) && key_avail!=FALSE)
    akt_event |= EV_KEY_AVAIL;  /* erwartete Taste betätigt */

   /*-- wenn auf die Bewegung der Maus gewartet wird, wird -*/
   /*-- das Ereignis nur akzeptiert, wenn die Maus in eine -*/
   /*-- andere Zeile und/oder Spalte des Textbildschirms   -*/
   /*-- bewegt wird.                                       -*/

   if (((wait_event & EV_MOU_MOVE) != 0) &&
        (spalte==moucol && zeile==mourow) )
    {         /* Maus bewegt, aber gleiche Bildschirmposition */
     akt_event &= (~EV_MOU_MOVE);      /* Move-Bit ausblenden */
     ende = (akt_event != 0);       /* bleiben Events übrig? */
    }
   else                               /* Ereignis eingetreten */
    ende = TRUE;
  }
 ev_col = moucol;                  /* aktuelle Mausposition und */
 ev_row = mourow;                  /* -bereich in globalen Vari- */
 ev_ber = mouber;                  /* ablen festhalten          */
 return( akt_event );               /* Event-Maske zurückliefern */
}

/***************************************************************
*  Funktion           : K b I S e t M o u s e H a n d l e r    *
**------------------------------------------------------------**
*  Aufgabe            : Setzt einen neuen Maus-Handler, der    *
*                       bei Eintritt eines bestimmten Ereig-   *
*                       nisses durch den Maustreiber aufge-    *
*                       rufen wird.                            *
*  Eingabe-Parameter: EVENT = Bit-Maske, die das Ereignis      *
*                             spezifiziert, bei dem der Maus-*
*                             Handler aufgerufen werden soll.*
*                     PTR   = Pointer auf den Maus-Handler     *
*  Return-Wert        : keiner                                 *
*  Info               : - EVENT kann durch Oder-Verknüpfung    *
*                         der verschiedenen Konstanten wie z.B.*
*                         EV_MOU_MOVE gebildet werden.         *
***************************************************************/

static void KbmISetMouHandler( unsigned event, MOUHAPTR ptr )
{
 union REGS regs;        /* Prozessorregs für Interruptaufruf */
 struct SREGS sregs; /* Segmentregister für Interruptaufruf */

 regs.x.ax = 0x000C;        /* Fktnr. für "Set Mouse Handler" */
 regs.x.cx = event;                       /* Event-Maske laden */
 regs.x.dx = FP_OFF( ptr );    /* Offsetadresse des Handlers */
 sregs.es = FP_SEG( ptr );    /* Segmentadresse des Handlers */
 MOUINTX( regs, regs, sregs );      /* Maustreiber aufrufen  */
}
```

Listing 2: *(Fortsetzung)*

```
/***************************************************************
*  Funktion           : K b m I G e t X                        *
**------------------------------------------------------------**
*  Aufgabe            : Ermittelt die Spalte, in der sich die  *
*                       Maus befindet.                         *
*  Eingabe-Parameter: keine                                    *
*  Return-Wert        : die Maus-Spalte in Bezug auf den Text- *
*                       Bildschirm                             *
***************************************************************/

static BYTE KbmIGetX( void )
{
 union REGS regs;        /* Prozessorregs für Interruptaufruf */
 regs.x.ax= 0x0003;      /* Fktnr.: für "Get mouse position" */
 MOUINT(regs, regs);                /* Maustreiber aufrufen  */
 return XTOCOL(regs.x.cx); /* Spalte umrechnen und zurückl. */
}

/***************************************************************
*  Funktion           : K b m I G e t Y                        *
**------------------------------------------------------------**
*  Aufgabe            : Ermittelt die Zeile, in der sich die   *
*                       Maus befindet.                         *
*  Eingabe-Parameter: keine                                    *
*  Return-Wert        : die Maus-Zeile in Bezug auf den Text-  *
*                       Bildschirm                             *
***************************************************************/

static BYTE KbmIGetY( void )
{
 union REGS regs;        /* Prozessorregs für Interruptaufruf */
 regs.x.ax= 0x0003;      /* Fktnr.: für "Get mouse position" */
 MOUINT(regs, regs);                /* Maustreiber aufrufen  */
 return YTOROW(regs.x.dx);  /* Zeile umrechnen und zurückl. */
}

/***************************************************************
*  Funktion           : K b m E n d                            *
**------------------------------------------------------------**
*  Aufgabe            : Beendet die Arbeit mit den Funktionen  *
*                       des KBM-Moduls.                        *
*  Eingabe-Parameter: keine                                    *
*  Return-Wert        : keiner                                 *
***************************************************************/

void KbmEnd( void )
{
 union REGS regs;        /* Prozessorregs für Interruptaufruf */

 /*-- wieder alte Interrupt-Handler installieren----------- */

 _dos_setvect( KB_INTR,  OldKbHandler );
 _dos_setvect( CTRLB_INT, OldCtrlBHandler );
 _dos_setvect( CTRLC_INT, OldCtrlCHandler );

 if ( makroanz )                     /* sind Makros definiert? */
  free( makvec );                /* Ja, Makro-Vektor freigeben */
 if ( fktanz )                  /* sind Funktionen definiert? */
  free( fktvec );           /* Ja, Funktions-Vektor freigeben */
 free( makrotab );           /* Bit-Tabellen wieder freigeben */

 MouHideMouse();                     /* Maus-Cursor ausblenden */
 regs.x.ax = 0;                    /* Reset des Maustreibers  */
 MOUINT(regs, regs);                 /* Maustreiber aufrufen  */
}

/***************************************************************
*  Funktion           : K b m I n i t                          *
**------------------------------------------------------------**
*  Aufgabe            : Leitet die Arbeit mit dem KBM-Modul    *
*                       ein.                                   *
*  Eingabe-Parameter: keine                                    *
*  Return-Wert        : TRUE, wenn eine Maus installiert ist,  *
*                       sonst FALSE                            *
*  Info               : Diese Funktion muß als erste Funktion  *
*                       aus diesem Modul aufgerufen werden.    *
***************************************************************/
```

Listing 2: *(Fortsetzung)*

```
BYTE KbmInit( void )
{
 union REGS regs;      /* Prozessorregs für Interruptaufruf */

 /*-- neuen Tastatur-Interrupt-Handler installieren --------*/

 OldKbHandler = _dos_getvect( KB_INTR );
 _dos_setvect( KB_INTR, KbdHandler );

 /*-- Ctrl-Break und Ctrl-C-Handler umleiten ---------------*/

 OldCtrlBHandler = _dos_getvect( CTRLB_INT );
 OldCtrlCHandler = _dos_getvect( CTRLC_INT );
 _dos_setvect( CTRLC_INT, BreakHandler );
 _dos_setvect( CTRLB_INT, BreakHandler );

 /*-- Puffer für die Makro- und die Funktionstabelle allok. */

 fkttab = ( makrotab = (BYTE *) malloc( 128 ) ) + 64;
 memset( makrotab, 0, 128 );  /* keine Makros, keine Fkt.s */

 atexit( KbmEnd );          /* am Programmende KbmEnd aufrufen */

 regs.x.ax = 0;                 /* Mouse-Treiber initialisieren */
 MOUINT(regs, regs);                      /* Maustreiber aufrufen */
 if ( regs.x.ax != 0xffff )     /* Maustreiber installiert? */
  return FALSE;                                        /* Nein */

 /*-- vertikale Auflösung des virtuellen Maus-Bildschirms --*/
 /*-- ermitteln                                            --*/

 tline = VioGetLines();          /* Anzahl Zeilen- und Spalten */
 tcol = VioGetCols();            /* im Textbildschirm ermitteln */

 /*-- Puffer für Maus-Bereiche allokieren und füllen -------*/

 bpuf = (BYTE *) malloc( tline * tcol );
 KbmIBufFill( 0, 0, tcol-1, tline-1, KEIN_BEREICH );

 moucol = KbmIGetX();              /* aktuelle Mausposition in */
 mourow = KbmIGetY();              /* globale Variablen laden  */
 KbmISetMouHandler( EV_MOU_ALL, MouHandler);
 return mavail = TRUE;                  /* Maus ist installiert */
}

/***********************************************************
*  Funktion          : M o u S h o w M o u s e              *
**--------------------------------------------------------**
*  Aufgabe           : Maus-Cursor auf dem Bildschirm an-   *
*                      zeigen.                              *
*  Eingabe-Parameter: keine                                 *
*  Return-Wert       : keiner                               *
***********************************************************/

void MouShowMouse( void )
{
 union REGS regs;      /* Prozessorregs für Interruptaufruf */

 regs.x.ax = 0x0001;              /* Fktnr.: für "Show Mouse" */
 MOUINT(regs, regs);                  /* Maustreiber aufrufen  */
}

/***********************************************************
*  Funktion          : M o u H i d e M o u s e              *
**--------------------------------------------------------**
*  Aufgabe           : Maus-Cursor vom dem Bildschirm ent-  *
*                      fernen.                              *
*  Eingabe-Parameter: keine                                 *
*  Return-Wert       : keiner                               *
***********************************************************/

void MouHideMouse( void )
{
 union REGS regs;      /* Prozessorregs für Interruptaufruf */

 regs.x.ax = 0x0002;              /* Fktnr. für "Hide Mouse" */
 MOUINT(regs, regs);                  /* Maustreiber aufrufen */
}
```

Listing 2: *(Fortsetzung)*

Probleme ergeben sich bereits daraus, daß der Maustreiber wichtige Informationen, wie die Event-Maske und die aktuelle Maus-Position in den verschiedenen Prozessorregister übergibt. Die C-Funktion muß deshalb über die Möglichkeit verfügen, auf diese Register zugreifen zu können, sie darf sie dabei aber nicht verändern, damit sie dem Maustreiber nach der Beendigung der Funktion unverändert zurückgeliefert werden. C-Funktionen aber verändern während ihrer Ausführung normalerweise zumindest die Register AX, BX, CX und DX und zerstören damit für den Maustreiber unter Umständen wichtige Informationen. Dazu kommt noch das Problem, daß sich beim Aufruf des Handlers im DS-Register nicht die Segmentadresse des C-Datensegments, sondern eine unvorhersehbare Adresse befindet und auf die globalen Variablen des C-Programms dadurch nicht zugegriffen werden kann.

Abhilfe schaffen hier zumindest partiell die Befehlswörter `_saveregs` und `_loadds`, die MSC ab der Version 5.1 bei der Deklaration von Funktionen unterstützt. Sie sorgen dafür, daß die jeweilige Funktion bei ihrem Aufruf den Inhalt der verschiedenen Prozessorregister auf dem Stack sichert und gleichzeitig die Segmentadresse des C-Datensegments in das DS-Register lädt. Bleibt nur noch das Problem zu lösen, wie die Inhalte der einzelnen Prozessorregister und damit die Event-Maske und die aktuelle Maus-Position ermittelt werden können.

Betrachtet man den Maschinen-Code, den MSC bei der Umsetzung einer C-Funktion erzeugt, zeigt sich, daß der Stack nicht nur zur Speicherung von Registerinhalten und Rücksprungadresse, sondern auch zur Speicherung der lokalen Variablen einer Funktion benutzt wird. Sie werden nach dem Funktionsaufruf unmittelbar an der Stelle auf dem Stack abgelegt, auf die das SP-Register (Stackpointer) zeigt. Normalerweise beginnen sie dadurch direkt hinter der Rücksprungadresse zum Aufrufer. Bei Verwendung des Befehlswortes `_saveregs` gehen ihnen jedoch noch die gesicherten Registerinhalte auf dem Stack voran (*Bild 1*).

Dadurch aber, daß man die Adresse einer (lokalen) Variablen über den &-Operator ermitteln kann und die Entfernung zwischen dieser Variablen und den einzelnen Registern bekannt ist, kann man auch auf die Register zugreifen, die auf dem Stack gesichert wurden. Ist `lova` die erste lokale Variable in einer `_saveregs`-Funktion, dann liefert der folgende Ausdruck den Inhalt des AX-Registers beim Aufruf der Funktion:

```
*((unsigned int*) &lova+10)
```

Lassen Sie sich bitte nicht dadurch verwirren, daß in dem obigen Ausdruck der Wert 10 als Entfernung zwischen der lokalen Variablen und dem AX-Register auf dem Stack angegeben wird, obwohl *Bild 1* zeigt, daß die Entfernung 20 und nicht 10 Byte beträgt. Da der obige Ausdruck jedoch nicht auf eine Variable vom Typ `byte`, sondern vom Typ `unsigned int` zugreift, muß der Abstand auch als ein Vielfaches der Größe dieses Datentyps angegeben werden:

```
( 10 * sizeof( unsigned int ) = 20 )
```

Auf diese Art und Weise verschafft sich der Maus-Handler `MouHandler()` bei seinem Aufruf Zugriff auf die Event-Maske, die das Ereignis wiederspiegelt, aufgrund dessen der Handler aufgerufen wurde. Darüber hinaus wird die aktuelle Mausposition aus den Registern CX und DX geladen. Da der Maustreiber die Mausposition intern immer in Bezug zu einem virtuellen Grafikbildschirm setzt, müssen diese Koordinaten zunächst in das Format des Textbildschirms umgerechnet werden, um für das Modul von Wert zu sein. Alle drei Informationen werden dabei in globalen Variablen festgehalten. Da der Maus-Handler bei jedem Ereignis im Zusammenhang mit der Maus aufgerufen wird, spiegeln diese Variablen immer die aktuelle Position der Maus und ihren Status wieder.

Eine weitere Aufgabe kommt dem Maus-Handler im Zusammenhang mit der Verwaltung der verschiedenen Bildschirmbereiche zu, auf die ich bei der Beschreibung des Unterschieds zwischen Maus und Tastatur bereits eingegangen bin. Zunächst ermittelt er aus der aktuellen Mausposition immer die Nummer des Bereichs, in dem sich die Maus befindet. (Bereiche werden über die Funktion `MouSetBereich()` definiert, die später noch vorgestellt wird.) Diese Information speichert er ebenfalls in einer globalen Variablen ab, wobei er jedoch zunächst überprüft, ob die Maus in einen neuen Bereich bewegt wurde. Da jeder Bereich dem Maus-Cursor ein eigenes Erscheinungsbild zuweisen kann, schaltet er in diesem Fall auf das Erscheinungsbild des neuen Bereichs um.

Damit schließt sich der Kreis und wir gelangen wieder zum Ursprung unserer Betrachtungen, der Funktion `KbmEventWait()`, zurück. Sie nämlich fragt die verschiedenen Ereignisse über die globalen Variablen ab, die durch den Maus- und Tastatur-Handler beeinflußt werden. So erkennt sie sofort den Eintritt des gewünschten Ereignisses und kann die Koordinaten der aktuellen Mausposition und die Nummer des Mausbereichs in spezielle globale Variablen laden, aus denen der Aufrufer diese Informationen später beziehen kann. Danach übergibt sie die Kontrolle wieder an den Aufrufer, der über die zurückgelieferte Ereignismaske feststellen kann, welches der erwarteten Ereignisse eingetreten ist.

Stellt der Aufrufer dabei z.B. fest, daß jetzt eine Taste bereitsteht, kann er diese über die Funktion `KbdGetKey()` abfragen, die den Code der Taste an den Aufrufer zurückliefert. Wie auch alle anderen Funktionen innerhalb des KBM-Moduls, die mit Tastencodes zu tun haben, wird der Tastencode dabei nicht in Form einer `char`-Variable, sondern als Datentyp `TASTE` zurückgeliefert, der einem `unsigned int` entspricht. Dadurch können die Codes der ASCII-Zeichen und die erweiterten Tastaturcodes der Cursor- und Steuertasten mit jeweils nur einer Variablen repräsentiert werden. Ein Tastencode mit einem Wert kleiner oder gleich 255 spiegelt dabei den Code eines ASCII-Zeichens wieder, während die Codes ab 256 den erweiterten Tastaturcodes vorbehalten sind.

```
/***************************************************************
* Funktion         : M o u S e t M o v e A r e a               *
**-----------------------------------------------------------**
* Aufgabe          : Definiert den Bildschirmbereich, in       *
*                    dem sich die Maus bewegen darf.           *
* Eingabe-Parameter: x1, y1 = Eckkoordinaten des Bild-         *
*                    x2, y2   schirmbereichs                   *
* Return-Wert      : keiner                                    *
* Info             : Befindet sich die Maus nicht in dem       *
*                    angegebenen Bildschirmbereich, wird       *
*                    sie automatisch dorthin versetzt.         *
***************************************************************/

void MouSetMoveArea( BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
 union REGS regs;          /* Prozessorregs für Interruptaufruf */

 regs.x.ax = 0x0008;       /* Fktnr. für "Set vertical Limits" */
 regs.x.cx = ROWTOY( ma_y1 = y1 );
 regs.x.dx = ROWTOY( ma_y2 = y2 );
 MOUINT(regs, regs);                    /* Maustreiber aufrufen */
 regs.x.ax = 0x0007;     /* Fktnr. für "Set horizontal Limits" */
 regs.x.cx = COLTOX( ma_x1 = x1 );
 regs.x.dx = COLTOX( ma_x2 = x2 );
 MOUINT(regs, regs);                    /* Maustreiber aufrufen */
}

/***************************************************************
* Funktion         : M o u P u s h P a r a                     *
**-----------------------------------------------------------**
* Aufgabe          : Sichert die aktuellen Maus-Parameter      *
*                    auf einem internen Stack                  *
* Eingabe-Parameter: keine                                     *
* Return-Wert      : keiner                                    *
* Info             : - zu den aktuellen Maus-Parametern        *
*                      zählen folgende Angaben:                *
*                        - die Mausbereiche                    *
*                        - der Standard-Maus-Cursor            *
*                        - die Maus-Area (Bewegungsbereich)    *
*                    - Der Aufrufer hat dafür Sorge zu tra-    *
*                      gen, daß der Stack nicht überläuft      *
*                      (nicht mehr als MAX_STACK-Einträge)     *
***************************************************************/

void MouPushPara( void )
{
 /*-- die verschiedenen Parameter auf dem Stack         ---------*/
 /*-- sichern und den Stack-Pointer inkrementieren ---------*/

 stackptr->anz_bereich = anz_bereich;
 stackptr->stdptr = stdptr;
 stackptr->akt_bereich = akt_bereich;
 stackptr->x1 = ma_x1;
 stackptr->x2 = ma_x2;
 stackptr->y1 = ma_y1;
 (stackptr++)->y2 = ma_y2;
}

/***************************************************************
* Funktion         : M o u P o p P a r a                       *
**-----------------------------------------------------------**
* Aufgabe          : Über die Funktionn MouPushPara ge-        *
*                    sicherte Maus-Parameter wieder vom        *
*                    internen Stack holen und aktivieren.      *
* Eingabe-Parameter: keine                                     *
* Return-Wert      : keiner                                    *
* Info             : - eine Übersicht über die Maus-Para-      *
*                      meter finden Sie bei MouPushPara.       *
*                    - Der Aufrufer muß dafür Sorge tragen,    *
*                      daß die Anzahl der Aufrufe von          *
*                      MouPushPara() und MouPopPara() aus-     *
*                      balanciert ist.                         *
***************************************************************/

void MouPopPara( void )
{
 --stackptr;                    /* den Stackpointer dekrmentieren */
```

Listing 2: *(Fortsetzung)*

```
 /*-- die verschiedenen Parameter über die jeweilige -------*/
 /*-- Funktion wieder restaurieren                    -------*/

 MouSetMoveArea( stackptr->x1, stackptr->y1,
                 stackptr->x2, stackptr->y2 );
 MouSetBereich(stackptr->anz_bereich, stackptr->akt_bereich);
 MouSetDefaultPtr( stackptr->stdptr );
}

/*************************************************************
* Funktion          : M o u S e t S p e e d                  *
**-------------------------------------------------------------**
* Aufgabe           : Setzt die Maus-Geschwindigkeit         *
* Eingabe-Parameter: - XSPEED = Geschwindigkeit in hori-     *
*                               zontaler Richtung            *
*                    - YSPEED = Geschwindigkeit in verti-    *
*                               kaler Richtung               *
* Return-Wert       : keiner                                 *
* Info              : - Beide Angaben beziehen sich auf die  *
*                       Einheit Mickey / Pixel. Erlaubt ist  *
*                       die Angabe von Werten zwischen 1 und *
*                       32767.                               *
*                     - Der Defaultwert für XSPEED ist 8,    *
*                       der für YSPEED 16.                   *
*************************************************************/

void MouSetSpeed( int xspeed, int yspeed )
{
 union REGS regs;         /* Prozessorregs für Interruptaufruf */

 regs.x.ax = 0x000f; /* Fktnr. "Set mickeys to pixel ratio" */
 regs.x.cx = xspeed;
 regs.x.dx = yspeed;
 MOUINT(regs, regs);                  /* Maustreiber aufrufen */
}

/*************************************************************
* Funktion          : M o u S e t P t r                      *
**-------------------------------------------------------------**
* Aufgabe           : Bewegt den Masu-Cursor an eine be-     *
*                     stimmte Bildschirmposition.            *
* Eingabe-Parameter: - COL = Bildschirmspalte                *
*                    - ROW = Bildschirmzeile                 *
* Return-Wert       : keiner                                 *
* Info              : - Beide Angaben beziehen sich auf den  *
*                       Textbildschirm.                      *
*************************************************************/

void MouSetPtr( int col, int row )
{
 union REGS regs;          /* Prozessorregs für Interruptaufruf */
 unsigned bereich;  /* Bereich, in den die Maus bewegt wird */

 regs.x.ax = 0x0004; /* Fktnr. "Set mouse pointer position" */
 regs.x.cx = COLTOX( moucol = col );
 regs.x.dx = ROWTOY( mourow = row );
 MOUINT(regs, regs);                  /* Maustreiber aufrufen */

 bereich = *(bpuf + mourow * tcol + moucol);  /* Ber. holen */
 if ( bereich != mouber )                /* neuer Bereich? */
  MouDefinePtr( (bereich == KEIN_BEREICH) ?            /* Ja */
        stdptr : (akt_bereich+bereich)->ptr_mask );
 mouber = bereich;   /* Bereichsnr. in globaler Var. merken */
}

/*************************************************************
* Funktion          : M o u D e f i n e P t r                *
**-------------------------------------------------------------**
* Aufgabe           : Legt die beiden Bit-Masken für die     *
*                     Erzeugung des Maus-Cursors fest.       *
* Eingabe-Parameter: MASK = die beiden Bit-Masken            *
* Return-Wert       : keiner                                 *
* Info              : - Der für MASK übergebene Wert sollte  *
*                       mit Hilfe des Makros MouPtrMask und  *
*                       der zugehörigen Konstanten aus der   *
*                       Include-Datei KBM.H erstellt werden. *
*************************************************************/
```

Listing 2: *(Fortsetzung)*

```
#pragma check_stack(off)        /* kein Stack-Checking hier */

void MouDefinePtr( PTRVIEW mask )
{
 union REGS regs;          /* Prozessorregs für Interruptaufruf */

 regs.x.ax = 0x000a;  /* Fktnr. für "Set text pointer type" */
 regs.x.bx = 0;                /* Software-Cursor einstellen */
 regs.x.cx = mask;                 /* Lo-Word ist AND-Maske */
 regs.x.dx = mask >> 16;           /* Hi-Word ist XOR-Maske */
 MOUINT(regs, regs);                /* Maustreiber aufrufen */
}

#pragma check_stack      /* alten Zustand im Hinblick auf das */
#pragma check_stack      /* Stack-Checking wieder herstellen  */

/*************************************************************
* Funktion          : M o u S e t D e f a u l t P t r        *
**-------------------------------------------------------------**
* Aufgabe           : Definiert das Erscheinungsbild des     *
*                     Maus-Cursors, wenn sich die Maus nicht *
*                     in einem spezifizierten Bereich be-    *
*                     findet.                                *
* Eingabe-Parameter: STANDARD = Bit-Maske für den Standard-  *
*                               Maus-Cursor                  *
* Return-Wert       : keiner                                 *
* Info              : - Der für MASK übergebene Wert sollte  *
*                       mit Hilfe der Makros PTRSAMECHAR     *
*                       etc., die in der Include-Datei KBM.H *
*                       definiert werden, erstellt werden.   *
*************************************************************/

void MouSetDefaultPtr( PTRVIEW standard )
{
 stdptr = standard;    /* Bit-Maske in globaler Var. merken */

 /*-- befindet sich die Maus im Augenblick in keinem Be- ---*/
 /*-- reich, wird der Maus-Cursor direkt auf das neue    ---*/
 /*-- Erscheinungsbild umgeschaltet.                     ---*/

 if ( MouGetBereich() == KEIN_BEREICH )
   MouDefinePtr( standard );
}

/*************************************************************
* Funktion          : K b d I P u t K e y                    *
**-------------------------------------------------------------**
* Aufgabe           : Bringt einen Tastencode an das Ende    *
*                     des internen Tastaturpuffers           *
* Eingabe-Parameter: KEY = der Tastencode                    *
* Return-Wert       : keiner                                 *
* Info              : Der Aufrufer ist dafür verantwortlich, *
*                     daß der Tastaturpuffer nicht überläuft.*
*************************************************************/

static void KbdIPutKey( TASTE key )
{
 *kbend++ = key;  /* eine Taste in den Tastaturpuffer laden */
 if (kbend == key_buf+KB_LEN) /* Ende des Puffers erreicht? */
  kbend = key_buf;          /* Ja, wieder am Anfang beginnen */
}

/*************************************************************
* Funktion          : K b d I N e x t K e y                  *
**-------------------------------------------------------------**
* Aufgabe           : Bringt einen Tastencode an den Anfang  *
*                     des internen Tastaturpuffers           *
* Eingabe-Parameter: KEY = der Tastencode                    *
* Return-Wert       : keiner                                 *
* Info              : Im Gegensatz zu KbdIPutKey hängt diese *
*                     Funktion den Tastencode nicht an das   *
*                     Ende des Tastaturpuffers an, sondern   *
*                     stellt ihn als nächsten zu lesenden    *
*                     Tastaturcode an den Anfang des Puffers.*
*************************************************************/
```

Listing 2: *(Fortsetzung)*

```
void KbdINextKey( TASTE key )
{
 if (--kbstart < key_buf)   /* Anfang des Puffers erreicht? */
  kbstart = key_buf+KB_LEN-1; /* Ja, auf Ende des P. setzen */
 *kbstart = key;  /* den Tastencode in den Tastaturp. laden */
 key_avail = TRUE; /* jetzt steht ein Zeichen zur Verfüfung */
}

/***************************************************************
* Funktion          : K b d I F i n d M a c                    *
**------------------------------------------------------------**
* Aufgabe           : Sucht ein Tastaturmakro innerhalb des    *
*                     Vektors mit den Makrobeschreibern.       *
* Eingabe-Parameter: KEY = Tastencode des Makros               *
* Return-Wert       : Pointer auf die Makro-Struktur           *
* Info              : Der Aufrufer sollte sich vor dem Auf-    *
*                     ruft dieser Funktion vergewissern,       *
*                     daß das Makro definiert ist. Nur in      *
*                     diesem Fall makht der Aufruf dieser      *
*                     Funktion Sinn.                           *
***************************************************************/

static KEY_MAK * KbdIFindMak( TASTE key )
{
 register KEY_MAK * lptr;              /* Laufzeiger in Vektor */

 /*-- den Vektor mit den Makrobeschreibern durchlaufen -----*/

 for (lptr = makvec; lptr->key != key; ++lptr )
  ;
 return lptr;                      /* Pointer zurückliefern */
}

/***************************************************************
* Funktion          : K b d I F i n d F k t                    *
**------------------------------------------------------------**
* Aufgabe           : Sucht die Verknüpfung zwischen einer     *
*                     Taste und einer Funktion im Vektor mit   *
*                     den Funktionsbeschreibern.               *
* Eingabe-Parameter: KEY = Tastencode der zugeord. Taste       *
* Return-Wert       : Pointer auf den Funktions-Beschreiber    *
* Info              : siehe KbdIFindMak                        *
***************************************************************/

static KEY_FKT * KbdIFindFkt( TASTE key )
{
 register KEY_FKT * lptr;              /* Laufzeiger in Vektor */

 /*-- den Vektor mit den Makrobeschreibern durchlaufen -----*/
 for (lptr = fktvec; lptr->key != key; ++lptr )
  ;
 return lptr;                      /* Pointer zurückliefern */
}

/***************************************************************
* Funktion          : K b d I G e t K e y                      *
**------------------------------------------------------------**
* Aufgabe           : Holt einen Tastencode aus dem inter-     *
*                     nen Tastaturpuffer.                      *
* Eingabe-Parameter: keine                                     *
* Return-Wert       : der Tastencode                           *
* Info              : Steht kein Tastencode zur Verfügung,     *
*                     wird der Wert 0xffff zurückgeliefert.    *
***************************************************************/

TASTE KbdIGetKey( void )
{
 register int retcode;

 if ( KbEmpty() )                     /* Tastaturpuffer leer? */
  return( 0xffff );                  /* Ja, mit Fehler zurück */

 /*- es steht eine Taste bereit ---------------------------*/
 retcode = *kbstart++;               /* einen Tastencode lesen */
 if (kbstart == key_buf+KB_LEN)     /* Ende des P. erreicht? */
  kbstart = key_buf;          /* Ja, wieder am Anfang beginnen */
 return retcode;             /* Tastencode zurückliefern */
}
```

Listing 2: *(Fortsetzung)*

Die Codes dieser Tasten werden in der Include-Datei `KBM.H` als Konstanten deklariert und können dadurch leicht in den Programmablauf eingebunden werden.

Die Include-Datei KBM.H

Die Include-Datei `KBM.H` ist der Schlüssel zum Zugriff auf die Funktionen des `KBM`-Moduls. In ihr finden sich neben den Typdefinitionen, die zur Arbeit mit den verschiedenen Funktionen benötigt werden, auch die Event-Codes für die verschiedenen Ereignisse und die Konstanten, die die erweiterten Tastaturcodes repräsentieren. Zusätzlich finden Sie hier die Deklarationen der verschiedenen Funktionen aus dem `KBM`-Modul und verschiedene Makros, die im folgenden vorgestellt werden.

MouGetCol(): Liefert die Bildschirmspalte, in der sich der Maus-Cursor bei der letzten Rückkehr aus `KbmEventWait( )` befand.

MouGetRow(): Liefert die Bildschirmzeile, in der sich der Maus-Cursor bei der letzten Rückkehr aus `KbmEventWait( )` befand.

MouGetBereich(): Liefert die Nummer des Bereichs, in dem sich der Maus-Cursor bei der letzten Rückkehr aus `KbmEventWait( )` befand. Wird hier der Wert der Konstanten `KEIN_BEREICH` zurückgeliefert, dann befand sich der Maus-Cursor in keinem der über `MouSetBereich( )` angegebenen Bildschirmbereiche.

IsAKey(): Liefert einen Wert ungleich 0 (true), wenn der übergebene Code den Code einer ASCII-Taste repräsentiert.

IsXKey(): Liefert einen Wert ungleich 0 (true), wenn der übergebene Code einen erweiterten Tastaturcode darstellt.

MouAvail(): Liefert TRUE, wenn eine Maus und ein zugehöriger Maustreiber installiert ist.

KEY(): Sollte zum Casten einer Variable vom Typ `char` in eine Variable vom Typ `TASTE` verwendet werden, da der Variablentyp `char` für MSC vorzeichenbehaftet ist und er bei einer automatischen Typumwandlung das Vorzeichen expandiert, was hier zu einem falschen Resultat führen kann.

MouGetAktCol(): Liefert die aktuelle Bildschirmspalte, in der sich der Maus-Cursor befindet.

MouGetAktRow(): Liefert die aktuelle Bildschirmzeile, in der sich der Maus-Cursor befindet.

IsFkt(): Liefert einen Wert ungleich 0 (true), wenn der angegebene Tastencode mit einer Funktion verknüpft ist, die bei der Betätigung dieser Taste aufgerufen wird. (Siehe dazu `KbdSetFkt( )`)

IsMakro(): Handelt es sich bei dem angegebenen Tastaturcode um den Code eines Tastaturmakros, liefert dieses Makro einen Wert ungleich 0 zurück (true). (Siehe dazu auch `KbmSetMakro( )`)

ELVEK(): Liefert die Anzahl der Elemente im angegebenen Vektor zurück.

MouPtrMask(): Erlaubt die Erstellung einer Bit-Maske für das Erscheinungsbild des Maus-Cursors, das z.B. beim Aufruf der Funktionen `MouDefinePtr()` und `MouSetDefaultPtr()` angegeben werden kann. Als erster Parameter muß diesem Makro entweder die Konstante `PTRSAMECHAR` oder die Konstante `PTRDIFCHAR()` übergeben werden. Als zweiter Parameter kommen die Konstanten `PTRSAMECOL`, `PTRINVCOL`, `PTRSAMECOLB`, `PTRINVCOLB`, `PTRDIFCOL()` und `PTRDIFCOLB()` in Frage.

MouSetMoveAreaAll(): Definiert den gesamten Bildschirm als den Bewegungsbereich der Maus, so daß sich der Maus-Cursor ungehindert bewegen kann.

KbdUngetKey(): Schiebt die angegebene Taste an den Anfang des internen Tastaturpuffers, so daß sie beim nächsten Aufruf von `KbdGetKey()` wieder zurückgeliefert wird.

Funktionen aus dem Modul KBM.C

Im folgenden werden die einzelnen Funktionen vorgestellt, die Ihnen das KBM-Modul zum Aufruf bereitstellt. Es geht dabei primär darum, die Aufgabe der jeweiligen Funktion und ihre Stellung innerhalb des Moduls als Ganzem zu beleuchten. Die zu übergebenden Parameter und ihre Bedeutung sowie die Art des Rückgabewerts entnehmen Sie bitte dem Listing des Moduls, das ausführlich dokumentiert ist und keine Fragen offen lassen sollte.

KbmInit(): Dies ist die Initialisierungsfunktion des Moduls, die als erste Funktion aus diesem Modul aufgerufen werden muß, um verschiedene Variablen zu initialisieren und die beiden Interrupt-Handler zu installieren. Die sie auf verschiedenen Funktionen aus dem VIO-Modul zugreift, das in der ersten Folge dieser Serie vorgestellt wurde, sollte diese Funktion erst nach der Initialisierung des VIO-Moduls, also nach dem Aufruf von `VioInit()`, aufgerufen werden.

KbmEnd(): Diese Funktion schließt die Arbeit mit dem KBM-Modul ab, indem sie die allokierten Puffer wieder freigibt und die installierten Handler für Maus und Tastatur wieder deaktiviert. Sie müssen diese Funktion allerdings nicht explizit aufrufen, da `KbmInit()` sie mit Hilfe der Bibliotheksfunktion `_atexit()` als eine Funktion festlegt, die bei der Beendigung des Programms automatisch aufgerufen werden soll.

KbmEventWait(): Diese Funktion wurde oben bereits besprochen. Sie erlaubt Ihnen, auf ein bestimmtes Ereignis im Hinblick auf die Maus oder die Tastatur zu warten.

MouSetBereich(): Mit Hilfe dieser Funktion können Sie die verschiedenen Bildschirmbereiche und die zugehörige Maske für den Maus-Cursor definieren.

MouShowMouse(): Bringt den Maus-Cursor auf den Bildschirm. Der Maustreiber verwaltet einen internen Zähler, mit dem er die Anzahl der Aufrufe der Funktionen `MouHideMouse()` und `MouShowMouse()` festhält. Die Maus wir nur angezeigt, bzw. vom Bildschirm entfernt, wenn die Anzahl dieser Aufruf ausgeglichen ist.

```
/***************************************************************
*  Funktion          : K b d G e t K e y                        *
**------------------------------------------------------------**
*  Aufgabe           : Holt eine Taste von der Tastatur.        *
*  Eingabe-Parameter: keine                                     *
*  Return-Wert       : Code der Taste                           *
*                      (siehe dazu Konstanten in KBM.H)         *
*  Info              : - Steht kein Zeichen bereit, wartet      *
*                        die Funktion auf die Betätigung        *
*                        einer Taste.                           *
***************************************************************/

TASTE KbdGetKey( void )
{
 TASTE key;
 KEY_MAK *makro;                 /* Pointer auf Makrobeschreiber */
 register TASTE *lptr;             /* Laufzeiger in Makropuffer */
 int i;                                    /* Schleifenzähler */

 while ( TRUE )                               /* Endlosschliefe */
  {
   while ( key_avail == FALSE ) /*warten, bis Taste betätigt*/
    ;
   /*-- alle Zeichen aus dem BIOS-Tastaturpuffer holen -----*/

   while ( _bios_keybrd( _KEYBRD_READY ) )  /*weitere Taste?*/
    {                                                 /* Ja */
     key = _bios_keybrd( _KEYBRD_READ );      /* Taste holen */
     KbdIPutKey( (key & 0xff) ? (key & 0xff) /* & in Puffer */
                            : key >> 8 | 0x100 );
    }
   key = KbdIGetKey();               /* Taste aus Puffer holen */
   if ( ( key & KEY_EXPANDED ) == 0 &&       /* Makroaufruf? */
        IsMakro( key ) )
    {                                                 /* Ja */
     makro = KbdIFindMak( key );       /* Beschreiber suchen */

     /*-- Makrozeichen in den internen Tastaturp. bringen --*/
     if ( (i = makro->anzahl - 1) != -1)  /* Länge nicht 0? */
      {                                             /* Nein */
       for ( lptr = makro->bufptr + i; /* das Makro durchl. */
             i--;
             lptr-- )
        KbdINextKey( *lptr | KEY_EXPANDED );
      key = *lptr;  /* erstes Makrozeichen direkt übergeben */
     }
    }

   key &= (~KEY_EXPANDED);             /* Makro-Bit ausblenden */
   key_avail = !KbEmpty();    /* Tastaturpuffer jetzt leer? */
   if ( IsFkt( key ) )                 /* Funktion aufrufen? */
    ( KbdIFindFkt( key )->fkt )();               /* Ja, do it */
   else                                               /* Nein */
    break;                 /* aus der Endlosschleife springen */
  }
 if ( lern && lblen )              /* im Aufzeichnungsmodus? */
  {                                                   /* Ja */
   *lbakt++ = key;    /* Taste im übergebenen Puffer merken */
   --lblen;   /* Anzahl verbleibender Tasten dekrementieren */
  }
 return key;          /* Tastencode an Aufrufer zurückliefern */
}

/***************************************************************
*  Funktion          : K b d F l u s h B u f                    *
**------------------------------------------------------------**
*  Aufgabe           : Löscht den internen und den BIOS-        *
*                      Tastatur-Puffer                          *
*  Eingabe-Parameter: keine                                     *
*  Return-Wert       : keiner                                   *
***************************************************************/

void KbdFlushBuf( void )
{
 /*-- alle Zeichen aus dem BIOS-Tastaturpuffer holen -------*/

 while ( _bios_keybrd( _KEYBRD_READY ) ) /* noch 'ne Taste? */
   _bios_keybrd( _KEYBRD_READ );            /* Ja, Taste holen */
```

Listing 2: *(Fortsetzung)*

```
 kbstart = kbend = key_buf;  /* kein Zeichen mehr im Puffer */
 key_avail = FALSE;                /* kein Zeichen verfügbar */
}

/**********************************************************
*  Funktion           : K b d C l e a r A l l M a c s     *
**------------------------------------------------------**
*  Aufgabe           : Löscht alle angelegten Tastatur-Makros.*
*  Eingabe-Parameter: keine                                 *
*  Return-Wert       : keiner                               *
**********************************************************/

void KBdClearAllMaks( void )
{
 if ( makroanz )          /* sind überhaupt Makros definiert? */
  {                                                   /* Ja */
   memset( makrotab, 0, 64 );           /* Bit-Tabelle löschen */
   free( makvec );     /* Vektor mit Makrobeschr. freigeben */
   makroanz = 0;              /* keine Makros mehr verfügbar */
  }
}

/**********************************************************
*  Funktion           : K b d D e l M a c r o             *
**------------------------------------------------------**
*  Aufgabe           : Löscht ein Tastaturmakro.           *
*  Eingabe-Parameter: - KEY = Tastencode der Makrotaste     *
*  Return-Wert       : keiner                               *
**********************************************************/

void KbdDelMakro( TASTE key )
{
 static KEY_MAK *makro;         /* Pointer auf Makro-Struktur */
 int i;

 if ( IsMakro( key ) )             /* ist das Makro definiert? */
  {                                                   /* Ja */
   if ( makroanz == 1 )      /* ist nur ein Makro definiert? */
    free( makvec );     /* Speicher des Makro-Vektors freig. */
   else                 /* es sind mehrere Makros definiert */
    {      /* die nachfolgenden Makrobeschreiber heranziehen */
     makro = KbdIFindMak( key );   /* Ja, Adresse ermitteln */
     i = makro - makvec;    /* Nummer des Elements im Vektor */
     memmove(makro, makro+1, (makroanz-i-1)*sizeof(KEY_MAK));
     makvec = realloc(makvec,
                      ( makroanz-1 ) * sizeof( KEY_MAK ));
    }
   --makroanz;              /* Anzahl der Makros dekrementieren */
   /*-- entsprechendes Bit in der Makro-Tabelle löschen ----*/
   *(makrotab + (key >> 3)) &= ~(1 << (key & 7));
  }
}

/**********************************************************
*  Funktion           : K b d S e t M a c r o             *
**------------------------------------------------------**
*  Aufgabe           : Definiert ein Tastaturmakro.        *
*  Eingabe-Parameter: - KEY    = Tastencode der Makrotaste  *
*                     - PTR    = Ptr. auf Vektor mit den um- *
*                               zusetzenden Tastencodes.   *
*                     - ANZAHL= Anzahl der Tasten im Vektor *
*  Return-Wert       : TRUE, wenn das Makro angelegt werden *
*                      konnte, sonst FALSE                  *
*  Info              : Während das Makro aktiv ist, darf der *
*                      Inhalt des übergebenen Vektors nicht  *
*                      verändert werden.                    *
**********************************************************/

BYTE KbdSetMakro( TASTE key, TASTE *ptr, BYTE anzahl )

{
 register KEY_MAK *makro;        /* Pointer auf Makro-Struktur */

 if ( IsMakro( key ) )        /* ist Makro bereits definiert? */
  makro = KbdIFindMak( key );       /* Ja, Adresse ermitteln */
 else                        /* Makro ist noch nicht definiert */
  {
```

Listing 2: *(Fortsetzung)*

```
   if ( makroanz )          /* ist schon ein Makro definiert? */
    {
     makro = realloc( makvec, (makroanz+1)*sizeof(KEY_MAK) );
     if ( makro )   /* konnte der Puffer vergrößert werden? */
      {                                               /* Ja */
       makvec = makro;           /* neue Vektoradresse merken */
       makro += makroanz++; /* Ptr auf neuen Beschr. setzen */
      }
    }
   else                                               /* Nein */
    {     /* zunächst Speicher für Makro-Vektor allokieren */
     makvec = makro = (KEY_MAK *) malloc( sizeof(KEY_MAK) );
     if ( makro )        /* konnte Speicher allokiert werden? */
      makroanz = 1;      /* Ja, jetzt ist ein Makro definiert */
    }
  }
 if ( makro )                                    /* alles o.k.? */
  {                                                   /* ja */
   makro->key = key;           /* Daten in die Struktur laden */
   makro->bufptr = ptr;
   makro->anzahl = anzahl;

   /*-- entsprechendes Bit in der Makro-Tabelle setzen -----*/
   *(makrotab + (key >> 3)) |= (1 << (key & 7));
   return TRUE;
  }
 else          /* es konnte kein Speicher allokiert werden */
   return FALSE;
}

/**********************************************************
*  Funktion           : K b d C l e a r A l l F k t       *
**------------------------------------------------------**
*  Aufgabe           : Löscht alle angelegten Verknüpfungen *
*                      zwischen einer Taste und einer Funk- *
*                      tion.                                *
*  Eingabe-Parameter: keine                                 *
*  Return-Wert       : keiner                               *
**********************************************************/
void KBdClearAllFkt( void )
{
 if ( fktanz )   /* sind überhaupt Verknüpfungen definiert? */
  {                                                   /* Ja */
   memset( fkttab, 0, 64 );             /* Bit-Tabelle löschen */
   free( fktvec ); /* Vektor mit Funktionsbeschr. freigeben */
   fktanz = 0;          /* keine Verknüpfungen mehr definiert */
  }
}

/**********************************************************
*  Funktion           : K b d D e l F k t                 *
**------------------------------------------------------**
*  Aufgabe           : Löscht die Verknüpfung zwischen einer *
*                      Taste und einer Funktion.            *
*  Eingabe-Parameter: - KEY = Tastencode der Taste          *
*  Return-Wert       : keiner                               *
**********************************************************/
void KbdDelFkt( TASTE key )
{
 static KEY_FKT *fkt;    /* Pointer auf Funktionsbeschreiber */
 int i;

 if ( IsFkt( key ) )       /* ist die Verknüpfung definiert? */
  {                                                   /* Ja */
   if ( fktanz == 1 ) /* ist nur eine Verknüpfung definiert?*/
    free( fktvec ); /* Ja, Speicher des Fkt.-Vektors freig. */
   else           /* es sind mehrere Verknüpfungen definiert */
    {      /* die nachfolgenden Fkt.beschreiber heranziehen */
     fkt = KbdIFindFkt( key ); /* Adresse Beschr. ermitteln */
     i = fkt - fktvec; /* Nummer des Beschreibers im Vektor */
     memmove(fkt, fkt+1, (fktanz-i-1)*sizeof(KEY_FKT));
     fktvec = realloc(fktvec, (fktanz-1) * sizeof( KEY_FKT ));
    }
   --fktanz;       /* Anzahl der Verknüpfungen dekrementieren */
   /*-- entsprechendes Bit in der Fkt.-Tabelle löschen -----*/
   *(fkttab + (key >> 3)) &= ~(1 << (key & 7));
  }
}
```

Listing 2: *(Fortsetzung)*

```
/***************************************************************
*  Funktion          : K b d S e t F k t                       *
**-----------------------------------------------------------**
*  Aufgabe           : Definiert die Verknüpfung zwischen      *
*                      einem Tastencode und einer Funktion,    *
*                      die beim Empfang dieses Tastencodes     *
*                      innerhalb der Funktion KbdGetKey auf-   *
*                      gerufen wird.                           *
*  Eingabe-Parameter: - KEY    = Tastencode                    *
*                      - FKTPTR= Pointer auf die aufzurufende  *
*                               Funktion.                      *
*  Return-Wert       : TRUE, wenn die Verknüpfung hergestellt  *
*                      wreden konnte, sonst FALSE              *
***************************************************************/

BYTE KbdSetFkt( TASTE key, FKTPTR fktptr )
{
 register KEY_FKT *fkt;        /* Pointer auf Fkt.Beschreiber */

 if ( IsFkt( key ) )   /* ist Verknüpfung bereits definiert? */
  fkt = KbdIFindFkt( key );          /* Ja, Adresse ermitteln */
 else                  /* Verknüpfung ist noch nicht definiert */
  {
   if ( fktanz )   /* ist schon eine Verknüpfung definiert? */
    {        /* Ja, Vektor mit Fkt-Beschreibern vergrössern */
     fkt = realloc( fktvec, (fktanz+1)*sizeof(KEY_FKT) );
     if ( fkt )      /* konnte der Puffer vergrößert werden? */
      {                                              /* Ja */
       fktvec = fkt;               /* neue Vektoradresse merken */
       fkt += fktanz++;      /* Ptr auf neuen Beschr. setzen */
      }
    }
   else                                            /* Nein */
    {        /* zunächst Speicher für Fkt-Vektor allokieren */
     fktvec = fkt = (KEY_FKT *) malloc( sizeof(KEY_FKT) );
     if ( fkt )          /* konnte Speicher allokiert werden? */
      fktanz = 1;      /* Ja, jetzt ist eine Verk. definiert */
    }
  }
 if ( fkt )                                    /* alles o.k.? */
  {                                                    /* ja */
   fkt->key = key;              /* Daten in die Stuktur laden */
   fkt->fkt = fktptr;

   /*-- entsprechendes Bit in der Fkt.-Tabelle setzen ------*/

   *(fkttab + (key >> 3)) |= (1 << (key & 7));
   return TRUE;
  }
 else            /* es konnte kein Speicher allokiert werden */
   return FALSE;
}


/***************************************************************
*  Funktion          : K b d L e a r n                         *
**-----------------------------------------------------------**
*  Aufgabe           : Startet die Aufzeichnung von Tasten-    *
*                      folgen.                                 *
*  Eingabe-Parameter: - LBTR = Pointer auf den Lern-Puffer,    *
*                              in den die Tastatur-Routine     *
*                              die Tastenfolge einträgt        *
*                      - LEN  = Länge des Lern-Puffers in      *
*                              TASTEn                          *
*  Return-Wert       : keiner                                  *
*  Info              : Wird die Aufzeichnung der Tastenfolge   *
*                      nicht vor dem Erreichen des Puffer-     *
*                      endes (LEN) durch den Aufruf von        *
*                      KbdStopLearn beendet, wird die Auf-     *
*                      zeichnung automatisch gestoppt.         *
***************************************************************/

void KbdLearn( TASTE * lbptr, int len )
{
 lernbuf = lbakt = lbptr;  /* Adresse des Puffers speichern */
 lblen = len;                  /* Länge des Puffers merken */
 lern = TRUE;  /* ab jetzt werden die Zeichen aufgezeichnet */
}
```

Listing 2: *(Fortsetzung)*

MouHideMouse(): Entfernt den Maus-Cursor vom Bildschirm. Siehe auch `MouShowMouse()`.

MouSetMoveArea(): Mit Hilfe dieser Funktion können Sie den Bildschirmbereich festlegen, innerhalb dessen sich der Maus-Cursor bewegen darf.

MouSetSpeed(): Diese Funktion erlaubt es Ihnen, die Mausgeschwindigkeit, d.h. das Verhältnis zwischen der Maus-Bewegung und der Bewegung des Maus-Cursors auf dem Bildschirm festzulegen.

MouDefinePtr(): Mit Hilfe dieser Funktion können Sie das aktuelle Erscheinungsbild des Maus-Cursors festlegen.

MouSetDefaultPtr(): Erlaubt Ihnen, das Erscheinungsbild des Maus-Cursors festzulegen, das vom Maus-Handler immer dann aktiviert wird, wenn sich die Maus in keinem der über `MouSetBereich()` festgelegten Bildschirmbereiche befindet.

MouSetPtr(): Gibt Ihnen die Möglichkeit, den Maus-Cursor an eine beliebige Bildschirmposition zu bewegen.

MouPushPara(): Sichert die aktuellen Maus-Parameter auf einem internen Stack des KBM-Moduls. Zu den Maus-Parametern zählen das Standard-Erscheinungsbild des Maus-Cursors, der erlaubte Bewegungsbereich sowie die verschiedenen Bildschirmbereiche, die über `MouSet-Bereich()` definiert wurden.

MouPopPara(): Holt die über `MouPushPara()` gesicherten Maus-Parameter wieder vom Stack und aktiviert sie.

KbdGetKey(): Liefert die jeweils nächste Taste und entfernt sie aus dem internen Tastaturpuffer.

KbdFlushBuf(): Löscht den internen Tastaturpuffer und alle Zeichen, die sich noch im BIOS-Tastaturpuffer befinden.

KbdClearAllMaks(): Löscht alle über `KbdSetMakro()` angelegten Tastaturmakros.

KbdClearAllFkt(): Löscht alle über `KbdSetFkt()` definierten Verknüpfungen zwischen einer Taste und einer Funktion.

KbdSetMakro(): Definiert ein Tastaturmakro. Bei der Betätigung der angegebenen Taste wird fortan immer die angegebene Tastensequenz in den internen Tastaturpuffer gebracht und beim Aufruf der Funktion `KbdGetKey()` Taste für Taste an den Aufrufer übergeben.

KbdDelMakro(): Löscht ein zuvor über `KbdSetMakro()` definiertes Tastaturmakro.

KbdSetFkt(): Mit Hilfe dieser Funktion können Sie erreichen, daß bei der Betätigung der angegebenen Taste jeweils die Funktion aufgerufen wird, deren Adresse Sie `KbdSet-Fkt()` übergeben haben. Dadurch wird eine Verknüpfung zwischen der Taste und der Funktion hergestellt.

KbdDelFkt(): Löscht eine zuvor über `KbdSetFKt()` hergestellte Verknüpfung zwischen einer Taste und einer Funktion.

KbdLearn(): Diese Funktion startet den sogenannten Lern-Modus, in dem alle eingegebenen Tasten in einem Puffer des Aufrufers aufgezeichnet werden.

KbdStopLearn(): Beendet den Lern-Modus und teilt dem Aufrufer mit, wieviele Tasten in dem Puffer aufgezeichnet wurden, dessen Adresse der Funktion `KbdLearn( )` übergeben wurde.

Das Demoprogramm KBMDEMO.C

Um Ihnen die Arbeit mit den Funktionen des KBM-Moduls auch an einem praktischen Beispiel zu demonstrieren, finden Sie in *Listing 3* das Demo-Programm KBMDEMO.C. Es besteht aus drei kleinen Demos, in denen Ihnen zunächst die Arbeit mit den Funktionen zum Zugriff auf die Tastatur verdeutlicht wird. Die zweite Demo zeigt dann, wie Sie mit Hilfe der Funktionen zur Abfrage der Maus und den verschiedenen Funktion aus dem VIO-Modul beliebige Bildschirmbereiche markieren können. Die abschließende Demo macht deutlich, daß Tastatur und Maus gleichzeitig abgefragt werden können und zeigt Ihnen die sinnvolle Verwendung verschiedener Mausbereiche.

Erstellen können Sie das Demo-Programm, indem Sie ihren Microsoft C-Compiler (Version 5.1) über den folgenden Befehl starten:

```
cl /AS kbmdemo.c kbm.c vio.c
```

Beachten Sie bitte, daß das KBM-Modul unter allen Speichermodellen des Microsoft C Compilers kompiliert werden kann, in einigen Speichermodellen bei der Kompilierung jedoch Warnmeldungen ausgegeben werden, mit denen der Compiler den Zugriff auf die Prozessorregister innerhalb des Maus-Handlers bemängelt. Diese Meldungen haben jedoch keinen Einfluß auf die Codegenerierung und können deshalb unbeachtet bleiben.

Michael Tischer

```
/***************************************************************
* Funktion          : K b d S t o p L e a r n                   *
**-----------------------------------------------------------**
* Aufgabe           : Beendet die Aufzeichnung der Tasten-     *
*                     folgen, die durch den Aufruf von          *
*                     KbdLearn initiiert wurde.                 *
* Eingabe-Parameter: keine                                      *
* Return-Wert       : Anzahl der aufgezeichneten Tasten         *
***************************************************************/

int KbdStopLearn( void )
{
 lern = FALSE;               /* Eingaben nicht mehr aufzeichnen */
 return lbakt - lernbuf;        /* Anzahl Tasten zurückliefern */
}
```

Listing 2: *(Ende)*

```
/***************************************************************/
/*                      K B M D E M O . C                      */
/*-------------------------------------------------------------*/
/*    Aufgabe        : Demonstriert die Arbeit mit den ver-    */
/*                     schiedenen Funktionen des KBM-Moduls.*/
/*-------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                         */
/*    entwickelt am  : 29.Ø1.1989                              */
/*    letztes Update :  2.Ø2.1988                              */
/*-------------------------------------------------------------*/
/*    Erstellung     : CL [Modell] KBMDEMO.C KBM.C VIO.C       */
/***************************************************************/

/*-- Include-Dateien einbinden ------------------------------*/

#include "vio.h"              /* für die Funktionen aus VIO.C */
#include "kbm.h"              /* für die Funktionen aus KBM.C */
#include <stdlib.h>                  /* für MIN() und MAX() */

/***************************************************************
* Funktionen im Zusammenhang mit der DEMO1                      *
***************************************************************/

BYTE lerne = FALSE,                      /* im Lern-Modus? */
     umsetzung = FALSE;            /* Umsetzung der Umlaute? */

void hilfe_demo1()
{
 static BYTE aktiv = FALSE;            /* rekursiver Aufruf? */
 static char *helpz[] =
  {
   "Diese Demo soll Ihnen  die Arbeit mit Tastaturmakros und",
   "der Verknüpfung von Tasten und Funktionen verdeutlichen.",
   "",
   "Sie können beliebige Zeichenfolgen  eingeben, die in dem",
   "jetzt  überdeckten Fenster erscheinen. Beendet  wird die",
   "Demo durch Betätigung der »Return«-Taste.",
   "",
   "Mit Hilfe  verschiedener Makros werden die deutschen Um-",
   "laute in ihre englische Schreibweise 'ae', 'oe' etc. um-",
   "gesetzt. Diese  Umsetzung können Sie mit  Hilfe der Tas-",
   "tenkombination »Alt U« an- und ausschalten.",
   "",
   "Die Aufzeichnung  beliebiger Tastenfolgen wird durch Be-",
   "tätigung von  »Ctrl L« eingeleitet und  bei erneuter Be-",
   "tätigung dieser Taste wieder abgestellt. Die aufgezeich-",
   "nete Tastenfolge kann dann über die Funktionstaste »F1Ø«",
   "abgerufen werden.",
   "",
   "                                  Bitte »Return« betätigen"
  };
 BYTE i,                                   /* Schleifenzähler */
      spalte, zeile;   /* nimmt Cursor-Spalte und Zeile auf */

 if ( !aktiv )                     /* Funktion bereits aktiv? */
  {                                                /* Nein */
   aktiv = TRUE;       /* die Help-Funktion ist jetzt aktiv */
   spalte = AKTS;                /* Cursor-Spalte ermitteln */
   zeile = AKTZ;                  /* Cursor-Zeile ermitteln */
   VioWinOpen( 6, 1, 66, 23 );              /* Fenster öffnen */
   VioFrame( VL(Ø), VO(Ø), VR(Ø), VU(Ø), EINRA, INVERS );
   VioClear( VL(1), VO(1), VR(-1), VU(-1), INVERS );
   for (i=Ø; i < ELVEK(helpz); ++i)   /* die Zeilen durchl. */
    VioPrint( VL(2), VO(2+i), INVERS, TRUE, helpz[i] );
   KbdFlushBuf();                 /* Tastaturpuffer löschen */
   while ( KbdGetKey() != CR )           /* auf RETURN warten */
    ;
   VioWinClose( TRUE );         /* Fenster wieder schliepen */
   VioSetCursor( spalte, zeile );      /* Cursor zuücksetzen */
   aktiv = FALSE; /* die Help-Funktion ist nicht mehr aktiv */
  }
}

/*-------------------------------------------------------------*/
void lernen()                    /* Lern-Modus an- ausschalten */
{
 static TASTE buf[1ØØ];                /* nimmt Tastenfolge auf */
 BYTE tlen;                            /* Länger der Tastenfolge */
```

Listing 3: `KBMDEMO.C`

```
 if ( lerne == FALSE )                              /* im Lern-Modus? */
  {                                       /* Nein, Starte den Lern-Modus */
   KbdDelMakro( F1Ø );                                 /* Makro löschen */
   KbdLearn( buf, ELVEK( buf ) );           /* Learn-Modus starten */
   VioPrint( VR(-7), VO(Ø), INVERS, FALSE, " LERN ");
   lerne = TRUE;                              /* Lern-Modus ist jetzt an */
  }
 else                                              /* Lernmodus anschalten */
  {
   tlen = KbdStopLearn();           /* Länge der Tastenfolge erm. */
   VioPrint( VR(-7), VO(Ø), NORMAL, FALSE, "——————");
   KbdSetMakro( F1Ø, buf, tlen );         /* Tastenafolge auf F1Ø */
   lerne = FALSE;                            /* Lern-Modus ist jetzt aus */
  }
}

/*-----------------------------------------------------------------*/
void umlaute()      /* Umsetzung der Umlaute an- ausschalten */
{
 /*-- Makros zur Umsetzung der Umlaute ----------------------*/
 static TASTE umlaut_k_ae[] = { KEY('a'), KEY('e') },
              umlaut_g_ae[] = { KEY('A'), KEY('E') },
              umlaut_k_oe[] = { KEY('o'), KEY('e') },
              umlaut_g_oe[] = { KEY('O'), KEY('E') },
              umlaut_k_ue[] = { KEY('u'), KEY('e') },
              umlaut_g_ue[] = { KEY('U'), KEY('E') },
              umlaut_sz[]   = { KEY('s'), KEY('s') };

 if ( umsetzung == FALSE )                  /* Umsetzung anschalten? */
  {
   /*-- die einzelnen Umlaute über die Makros umdefinieren -*/
   KbdSetMakro( KEY('ä'), umlaut_k_ae, ELVEK( umlaut_k_ae ) );
   KbdSetMakro( KEY('Ä'), umlaut_g_ae, ELVEK( umlaut_g_ae ) );
   KbdSetMakro( KEY('ö'), umlaut_k_oe, ELVEK( umlaut_k_oe ) );
   KbdSetMakro( KEY('Ö'), umlaut_g_oe, ELVEK( umlaut_g_oe ) );
   KbdSetMakro( KEY('ü'), umlaut_k_ue, ELVEK( umlaut_k_ue ) );
   KbdSetMakro( KEY('Ü'), umlaut_g_ue, ELVEK( umlaut_g_ue ) );
   KbdSetMakro( KEY('ß'), umlaut_sz, ELVEK( umlaut_sz ) );
   umsetzung = TRUE;                          /* Umsetzung jetzt an */
  }
 else                          /* Umsetzung ist an, jetzt ausschalten */
  {                                   /* Makro-Definitionen wieder löschen */
   KbdDelMakro( KEY('ä') );
   KbdDelMakro( KEY('Ä') );
   KbdDelMakro( KEY('ö') );
   KbdDelMakro( KEY('Ö') );
   KbdDelMakro( KEY('ü') );
   KbdDelMakro( KEY('Ü') );
   KbdDelMakro( KEY('ß') );
   umsetzung = FALSE;                           /* Umsetzung jetzt aus */
  }
}

/*-----------------------------------------------------------------*/

void demo1()
{
 TASTE key;
 BYTE i, j;

 umlaute();                        /* Umsetzung der Umlaute anschalten */
 KbdSetFkt( ALT_U, umlaute); /*Umsch. der Umlaute über ALT_U*/
 KbdSetFkt( F1, hilfe_demo1 );                /* F1 liefert Infos */
 KbdSetFkt( CTRL_L, lernen ); /*CTRL_L = Tastensequenz aufz.*/

 VioWinOpen( VL(Ø), VO(Ø), VL(2Ø), VU(Ø) );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), NORMAL );
 VioFrame( VL(Ø), VO(Ø), VR(Ø), VU(Ø), DOPRA, NORMAL );
 VioPrint( VL(2), VU(Ø), INVERS, FALSE, "F1 = Hilfe " );
 VioSetCursor( VL(2), VU(-1) );
 do
  {
   key = KbdGetKey();                                  /* Taste holen */
   if ( IsAKey( key ) )                              /* ASCII-Zeichen? */
    VioPrintf( VL(2), VU(-1), NORMAL, FALSE, "%c", key );
   else             /* erweiterten Tastaturcode empfangen */
    VioPrintf( VL(2), VU(-1), NORMAL, FALSE,
              "EXTENDED: %3d", key );
```

Listing 3: *(Fortsetzung)*

```
   VioScrollUp( VL(1), VO(2), VR(-1), VU(-1), 1, NORMAL );
  }
 while ( key != CR );          /* wiederholen bis RETURN betätig */
 if ( lerne )                                /* Lern-Modus noch an? */
  lernen();                                        /* Ja, ausschalten */
 if ( umsetzung )        /* Umsetzung der Umlaute noch aktiv? */
  umlaute();             /* Ja, Umleitung wieder abschalten */
 KbdDelFkt( ALT_U );               /* die Verknüpfung zwischen den */
 KbdDelFkt( F1 );                 /* verschiedenen Tasten und den */
 KbdDelFkt( CTRL_L );             /* Funktionen wieder auflösen   */
 KbdFlushBuf();                          /* Tastatur-Puffer löschen */
 VioWinClose( TRUE );               /* Fenster wieder schließen */
}

/**********************************************************
* Funktionen im Zusammenhang mit der DEMO2                *
**********************************************************/

void demo2( void )
{
 static char *info[] =
  {
   "Innerhalb der umrahmten Fläche können"
   " Sie den Maus-Cursor  beliebig",
   "bewegen. Bei fortwährender Betätigung "
   "des linken Maus-Knopfes können",
   "Sie einen  Bildschirmbereich markieren"
   ", der dabei farblich hervorge-",
   "hoben wird.",
   "Beenden Sie diese Demo durch Niederdrü"
   "cken des rechten Mausknopfes."
  };

 TASTE key = KEY(Ø);
 BYTE row1, col1,                    /* Ursprungsposition der Maus */
      row2, col2,                         /* aktuelle Mausposition */
      newrow, newcol,               /* jeweils neue Mausposition */
      deltax, deltay,          /* Entfernung vom Ursprungspunkt */
      i,                                                  /* Dummy */
      sektor;   /* Sektor, in dem sich der Maus-Cursor bef. */
 int event;              /* Bit-Maske für jeweiliges Ereignis */

 VioHideCursor();                              /* Cursor verstecken */
 VioColor( VL(Ø), VO(Ø), VR(Ø), VO(4), INVERS );
 for (i=Ø; i < ELVEK(info); ++i)           /* die Zeilen durchl. */
  VioPrint( VL(5), VO(i), INVERS, TRUE, info[i] );
 VioFrame( VL(Ø), VO(5), VR(Ø), VU(Ø), EINRA, NORMAL );
 MouSetMoveArea( VL(1), VO(6), VR(-1), VU(-1) );
 MouSetPtr( VCOL >> 1, ((VROW-5) >> 1) + 5 );
 MouShowMouse();                           /* Maus-Cursor anzeigen */
 while ( TRUE )                                  /* Endlosschleife */
  {
   MouSetDefaultPtr( MouPtrMask( PTRSAMECHAR, PTRINVCOL ) );

   /*-- warten, bis der linke oder der rechte Mausknopf ----*/
   /*-- niedergedrückt wird                               ----*/
   event = KbmEventWait( EV_LEFT_PRESS  |
                         EV_RIGHT_PRESS | EV_KEY_AVAIL );

   if ( event & EV_KEY_AVAIL )                  /* Taste betätigt? */
    {                                /* Ja, wurde ESCAPE betätigt? */
     if ( (key = KbdGetKey()) == ESC )
      break;                                     /* Ja, Demo beenden */
    }
   if ( event & EV_LEFT_PRESS ) /*linke Maustaste betätigt? */
    {                                                         /* Ja */
     MouSetDefaultPtr(MouPtrMask( PTRSAMECHAR, PTRSAMECOL ));
     row1 = row2 = MouGetRow();   /* aktuelle Mauszeile und */
     col1 = col2 = MouGetCol();   /* -spalte merken         */
     deltax = deltay = sektor = Ø;
     MouHideMouse();                      /* Maus-Cursor ausblenden */
     VioPrint( col1, row1, INVERS, FALSE, " " );
     MouShowMouse();           /* Maus-Cursor wieder einblenden */
     while( TRUE )                               /* Endlosschleife */
      {
       /*-- warten bis die Maus bewegt oder der linke -----*/
       /*-- Mausknopf wieder losgelassen wird          -----*/
       event = KbmEventWait( EV_MOU_MOVE | EV_LEFT_REL );
```

Listing 3: *(Fortsetzung)*

```
      if ( event & EV_MOU_MOVE )            /* Maus bewegt? */
       {                                            /* Ja */
        newrow = MouGetRow();     /* aktuelle Position des  */
        newcol = MouGetCol();     /* Maus-Cursors ermitteln */
        MouHideMouse();           /* Maus-Cursor wegblenden */
        if ( sektor != ( i = (newcol>col1? 1:0) +
                             (newrow>row1? 2:0)))
         {                                 /* neuer Sektor */
          VioColor(min(col1, col2), min(row1, row2),
                   max(col1, col2), max(row1, row2), NORMAL);
          col2 = newcol;          /* neue Position updaten */
          row2 = newrow;
          VioColor(min(col1, col2), min(row1, row2),
                   max(col1, col2), max(row1, row2), INVERS);
          sektor = i;               /* neuen Sektor merken */
          deltax = abs( newcol - col1 ); /* Entfernung vom */
          deltay = abs( newrow - row1 ); /* Ursprung ber.  */
         }
        else                          /* kein neuer Sektor */
         {
          if ( newcol != col2 )             /* neue Spalte? */
           {                                         /* Ja */
            /*-- gehören jetzt mehr Spalten zum Bereich? --*/
            if (deltax < (i=abs(newcol-col1)))
             VioColor( (newcol < col2) ? newcol : col2+1,
                       min(row1, row2),
                       (newcol > col2) ? newcol : col2-1,
                       max(row1, row2), INVERS );
            else  /* weniger Spalten im markierten Bereich */
             VioColor( (newcol < col2) ? newcol+1 : col2,
                        min(row1, row2),
                        (newcol > col2) ? newcol-1 : col2,
                        max(row1, row2), NORMAL );
            deltax = i;        /* neue Anzahl Spalten merken */
           }
          if ( newrow != row2 )              /* neue Zeile? */
           {                                         /* Ja */
            /*-- gehören jetzt mehr Zeilen zum Bereich? ---*/
            if ( deltay < (i=abs(newrow-row1)) )
             VioColor( min(col1, newcol),
                       (newrow < row2) ? newrow : row2+1,
                       max(col1, newcol),
                       (newrow > row2) ? newrow : row2-1,
                       INVERS );
            else                 /* weniger Zeilen im Bereich */
             VioColor( min(col1, newcol),
                       (newrow < row2) ? newrow+1 : row2,
                       max(col1, newcol),
                       (newrow > row2) ? newrow-1 : row2,
                       NORMAL );
            deltay = i;        /* neue Anzahl Zeilen merken */
           }
          col2 = newcol;             /* neue Position merken */
          row2 = newrow;
         }
        MouShowMouse();  /* Maus-Cursor wieder einblenden */
       }
      else         /* linker Mausknopf wurde losgelassen */
       {
        MouHideMouse();           /* Maus-Cursor ausblenden */
        VioColor(min(col1, col2), min(row1, row2),
                 max(col1, col2), max(row1, row2), NORMAL);
        MouShowMouse();  /* Maus-Cursor wieder einblenden */
        break;               /* aus der Schleife ausbrechen */
       }
     }
   }
  else             /* der rechte Mausknopf wurde betätigt */
   break;             /* aus der Endlosschleife ausbrechen */
 }

 if ( key != ESC )   /* wurde die Demo über ESCAPE beendet? */
  /*-- Nein, über den rechten Maus-Knopf -----------------*/
  KbmEventWait( EV_RIGHT_REL );     /* auf Loslassen warten */
 MouHideMouse();                    /* Maus-Cursor ausblenden */
 VioClearScreen( NORMAL );          /* den Bildschirm löschen */
 MouSetMoveAreaAll();  /*Bewegungsfreiheit wieder herstellen*/
}
```

Listing 3: *(Fortsetzung)*

```
/**************************************************************
* Funktionen im Zusammenhang mit der DEMO3                    *
**************************************************************/
void hilfe()
{
 static char *hz[] =
  {
   "Bewegen Sie den Zeichenblock mit Hilfe der Cursor-",
   "Tasten oder indem Sie den Maus-Cursor in den Bild-",
   "schirmrahmen bewegen und dann den linken Mausknopf",
   "niederdrücken.",
   "",
   "Beenden  Sie die Demo mit Hilfe des  rechten Maus-",
   "knopfes.",
   "                                                 ",
   "                                   ┌─────────┐  ",
   "                                   │   O K   │  ",
   "                                   └─────────┘  "
  };
 BEREICH bereich;
 BYTE i,                                 /* Schleifenzähler */
      spalte,                        /* aktuelle Mausposition */
      zeile;

 spalte = MouGetAktCol();           /* aktuelle Mausspalte holen */
 zeile = MouGetAktRow();             /* aktuelle Mauszeile holen */
 MouPushPara();                      /* Mausparameter sichern */
 VioWinOpen( 15, 5, 69, 18 );                 /* Fenster öffnen */
 VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, INVERS );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), INVERS );
 for (i=0; i < ELVEK(hz); ++i)          /* die Zeilen durchl. */
  VioPrint( VL(2), VO(2+i), INVERS, FALSE, hz[i] );
 MouSetMoveArea( VL(1), VO(1), VR(-1), VU(-1) );
 MouSetDefaultPtr( MouPtrMask(PTRSAMECHAR, PTRINVCOL) );
 bereich.x1 = VR(-15);        /* Position des OK-Felds setzen */
 bereich.y1 = VU(-4);
 bereich.x2 = VR(-3);
 bereich.y2 = VU(-2);
 bereich.ptr_mask = MouPtrMask( PTRDIFCHAR( 251 ),
                                PTRDIFCOLB( 0x70 ) );
 MouSetBereich( 1, &bereich );       /* das OK-Feld definieren */
 MouSetPtr( VL(4), VU(-2) );             /* Maus-Cursor setzen */
 MouShowMouse();               /* Maus-Cursor wieder einblenden */
 do {         /* auf Betätigung des linken Mausknopfs warten */
  KbmEventWait( EV_LEFT_PRESS );
 }
 while ( MouGetBereich() != 0 ); /* warten bis Maus im OK-F.*/
 KbmEventWait( EV_LEFT_REL );
 MouHideMouse();                      /* Maus-Cursor wegblenden */
 VioWinClose( TRUE );              /* Fenster wieder schließen */
 MouPopPara();                    /* Mausparameter zurückholen */
 MouSetPtr( spalte, zeile );       /* Maus zurück an alte Pos. */
}

/*----------------------------------------------------------*/
void demo3( void )
{
 static BEREICH bereiche[5] =
  {
   {0, 0, 79, 0, MouPtrMask(PTRDIFCHAR(0x18), PTRINVCOLB)   },
   {0, 1, 0, 23, MouPtrMask(PTRDIFCHAR(0x1b), PTRINVCOLB)   },
   {0, 24, 79, 24, MouPtrMask(PTRDIFCHAR(0x19), PTRINVCOLB) },
   {79, 1, 79, 23, MouPtrMask(PTRDIFCHAR(0x1a), PTRINVCOLB) },
   {5, 24, 11, 24, MouPtrMask(PTRSAMECHAR, PTRINVCOL)       },
  };
 TASTE key;
 int ev,                     /* Bit-Maske des jeweiligen Events */
     richtung;     /* Richtung, in der gescrollt werden soll */
 BYTE spalte = (VCOL >> 1) - 2,
      zeile = (VROW >> 1) - 2,
      ir;       /* TRUE, wenn die Maus sich im Rahmen befindet */
 if ( VCOL!=80 || VROW!=25 )  /* nicht im 80*25 Bildschirm? */
  {                   /* nein, Koordinaten in BEREICHE laden */
   bereiche[0].x2 = bereiche[2].x2 =
   bereiche[3].x1 = bereiche[3].x2 = VR(0);
   bereiche[2].y1 = bereiche[2].y2 =
   bereiche[4].y1 = bereiche[4].y2 = VU(0);
   bereiche[1].y2 = bereiche[3].y2 = VU(-1);
  }
```

Listing 3: *(Fortsetzung)*

```
MouSetDefaultPtr( MouPtrMask( PTRSAMECHAR, PTRINVCOL ) );
VioHideCursor();                /* blinkenden Cursor ausblenden */
VioClearScreen( NORMAL );                /* Bildschirm löschen */
VioFrame( VL(Ø), VO(Ø), VR(Ø), VU(Ø), DOPRA, NORMAL );
VioPrint( VL(4), VU(Ø), NORMAL, FALSE, "╡        ╞");
VioPrint( VL(5), VU(Ø), INVERS, FALSE, " HILFE ");
VioFill( spalte, zeile, spalte+4, zeile+4, '▪', NORMAL );
MouShowMouse();                      /* Maus-Cursor anzeigen */
MouSetBereich( 5, bereiche );    /* 4 Mausbereiche anlegen */
while ( TRUE )                             /* Endlosschleife */
 {
  ev = KbmEventWait( EV_RIGHT_PRESS |
                     EV_LEFT_PRESS  |
                     EV_KEY_AVAIL     );
  if ( ev & EV_RIGHT_PRESS )  /*rechter Mausknopf gedrückt?*/
   break;                       /* Ja, die Schleife beenden */

  if ( ev & EV_KEY_AVAIL )               /* Taste betätigt? */
   {      /* Ja, testen ob Demo durch ESCAPE beendet wird */
    if ( (key = KbdGetKey()) == ESC )
     break;                            /* Ja, Demo beenden */
    switch (  key  )                    /* Taste auswerten */
     {
      case CUP     : richtung = Ø; break;      /* Cursor Up */
      case CLEFT   : richtung = 1; break;    /* Cursor Left */
      case CDOWN   : richtung = 2; break;    /* Cursor Down */
      case CRIGHT  : richtung = 3; break;   /* Cursor Right */
      default      : richtung = KEIN_BEREICH;
     }
   }
  else                 /* linker Maus-Knopf niedergedrückt */
   richtung = MouGetBereich();          /* Maus-Bereich holen */
  if ( richtung != KEIN_BEREICH )              /* Scrolling? */
   {                                                  /* Ja */
    if ( MouGetAktBer() == KEIN_BEREICH )  /* akt. Bereich */
     {      /* Maus ist innerhalb des Bildschirm-Rahmens */
      ir = TRUE;
      MouHideMouse();                /* Maus-Cursor ausblenden */
     }

    switch ( richtung )  /* die Richtung bestimmt den Code */
     {                   /* nicht das Bewußtsein!?         */
      case Ø : if ( zeile != VO(1) )
                {            /* nocht nicht in erster Zeile */
                 VioScrollUp( VL(1), VO(2), VR(-1),
                              VU(-1), 1, NORMAL );
                 --zeile;
                }
               break;
```

Listing 3: *(Fortsetzung)*

```
      case 1 : if ( spalte != VL(1) )
                {           /* noch nicht in erster Spalte */
                 VioScrollLeft( VL(2), VO(1), VR(-1),
                                VU(-1), 1, NORMAL );
                 --spalte;
                }
               break;
      case 2 : if ( zeile+4 != VU(-1) )
                {           /* noch nicht in letzter Zeile */
                 VioScrollDown( VL(1), VO(1), VR(-1),
                                VU(-2), 1, NORMAL );
                 ++zeile;
                }
               break;
      case 3 : if ( spalte+4 != VR(-1) )
                {          /* noch nicht in letzter Spalte */
                 VioScrollRight( VL(1), VO(1), VR(-2),
                                 VU(-1), 1, NORMAL );
                 ++spalte;
                }
               break;
      case 4 : hilfe();              /* Hilfsfunktion aufrufen */
     }
    if ( ir )                       /* war Maus im Rahmen? */
     MouShowMouse();     /* Ja, Maus-Cursor wieder anzeigen */
   }
 }
 MouHideMouse();                     /* Maus-Cursor ausblenden */
}


/**************************************************************
* H A U P T P R O G R A M M                                   *
**************************************************************/

void main()
{
 VioInit();                       /* Vio-Modul initialisieren */
 KbmInit();                       /* Kbm-Modul initialisieren */
 VioClearScreen( NORMAL );           /* den Bildschirm löschen */
 demo1();                /* die verschiedenen Demos ausführen */
 if ( MouAvail() )             /* ist eine Maus.installiert? */
  {                                                     /* Ja */
   demo2();                      /* die Mausdemos ausführen */
   demo3();
  }
 VioClearScreen( NORMAL );          /* den Bildschirm löschen */
 VioSetCursor( Ø, Ø );       /* Cursor in obere linke B.Ecke */
}
```

Listing 3: *(Ende)*

Grundlagen der Farbmischung:

Der Einsatz von Farbe im Raster-Video-Modell

In den letzten Jahren fand eine dramatische Revolution in der Computergrafik statt. Als ich begann, am Computer zu arbeiten, war ich mit einem kleinen Monochrom-Monitor ohne Grafikmöglichkeiten mit 80 Zeichen und 25 Zeilen zufrieden. Heute fühle ich mich schon bei einer Auflösung von weniger als 640 x 480 Pixel eingeschränkt. Heutzutage sind Bildschirme mit einer Auflösung von 1280 x 900 keine Seltenheit und einige haben eine noch höhere Auflösung. Früher gaben wir uns mit monochromer Ausgabe zufrieden. Nunmehr erwarten wir zumindest 8 oder 16 Farben, selbst vom einfachsten Bildschirmsystem.

Die Einführung von Farbgrafik auf Personalcomputern hatte einen tiefgreifenden Einfluß auf die Software-Entwicklung. Desktop-Publishing-Systeme, Programme für Präsentationsgrafiken, Zeichenprogramme und Bildverarbeitungstools wurden teilweise durch diese Technologie erst ermöglicht. Trotz dieser Fortschritte nehmen wir noch immer die Möglichkeiten der Farbverarbeitung auf dem Computer als gegeben hin. Wie eine Analyse zeigt, sind selbst die besten heutigen Bildsysteme noch weit von der Wirklichkeit entfernt.

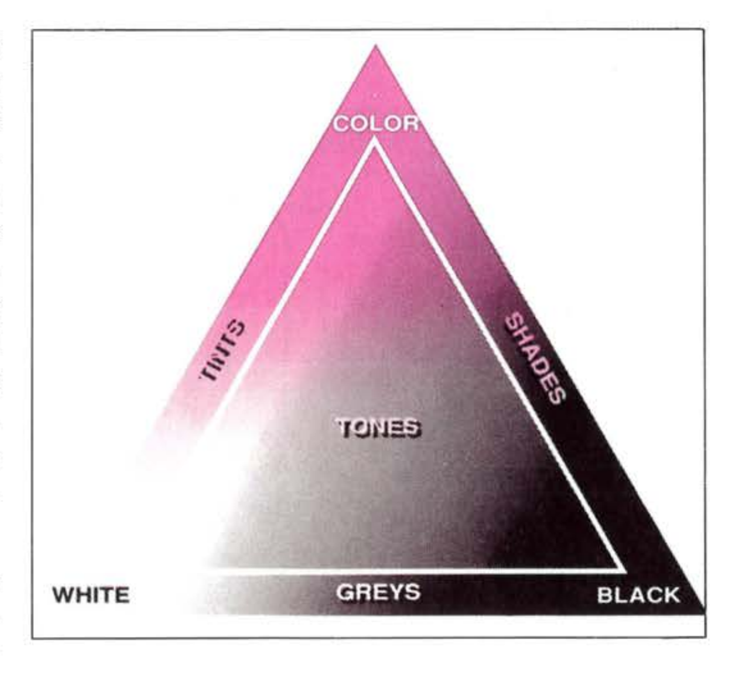


Bild 1: *Einfärbungen (tints), Schattierungen (shades) und Farbtöne (tones).*

Farbzusammensetzung

Farbempfinden ist sehr subjektiv und hängt ganz von unseren Wahrnehmungen ab. Die Farbe, die wir sehen, setzt sich aus den physikalischen Eigenschaften des jeweiligen Objekts, den Lichtquellen, die dieses Objekt beleuchten und den physiologischen und wahrnehmungsbedingten Eigenschaften des menschlichen Auges zusammen.

Der Prozeß der Wahrnehmung ist sehr komplex und keine einzige physikalische Theorie hat bis jetzt auch nur die einfachste Beobachtung zufriedenstellend erklären können. Beispielweise bemerken wir, vom physikalischen Standpunkt aus gesehen, subtilste Unterschiede in der Intensität von Farben; die Fähigkeit, kleine Differenzen in der Färbung oder von einer Farbe zur anderen zu erkennen, variiert jedoch stark. Tatsächlich erscheinen viele Energieniveaus des Lichtes, praktisch gesehen, gleich. Der Mensch versucht seit Jahrhunderten, diese physikalischen Beobachtungen zu deuten und mit unserem Farbempfinden in Einklang zu bringen. Zwei generelle Beschreibungsmethoden sind aus diesen Forschungen hervorgegangen.

Die erste Methode ist die von Künstlern verwendete: Einfärbungen, Schattierungen und Farbtöne (siehe *Bild 1*). Dieses subtraktive Modell ergibt sich aus der Art, in der ein Künstler ein Bild auf weißes Papier malt. Farbschichten werden solange aufeinander gelegt, bis der gewünschte Effekt erzielt ist. Weiß wird einfach durch Weglassen aller Pigmente, Schwarz durch Kombination aller Pigmente erzielt. Eine Einfärbung wird durch Verminderung der Sättigung einer Grundfarbe erreicht oder, anders gesagt, durch Mischen mit Weiß. Schattierungen kommen dadurch zustande, daß man eine Grundfarbe mit Schwarz mischt, d.h. ihre Helligkeit reduziert. Der Farbton schließlich resultiert aus der Mischung einer Grundfarbe sowohl mit Weiß, als auch mit Schwarz. Die Kombination dieser Verfahren produziert verschiedene Farben derselben Farbabtönung in unterschiedlichen Helligkeits- und Sättigungsgraden.

Bei der zweiten Methode geht es um Farbabtönung, Sättigung und Helligkeit. Von einem beschreibenden Standpunkt fügt dies eine weitere Dimension zu der traditionellen Einfärbungs-, Schattierungs- und Farbton-Beschreibungsmethode, wie von Künstlern verwendet, hinzu. Die Farbabtönung bezeichnet die Farbmenge einer bestimmten Farbe wie Rot, Grün oder Blau. Die Sättigung ist insofern schwieriger, als sie sich auf die Reinheit der Farbe bezieht, d.h., wie stark sie erscheint. Mischt man beispielsweise reines rotes Licht mit weißem erhält man Rosa. Dieses Rosa ist wie Rot, nur weniger gesättigt. Die Helligkeit schließlich bezieht sich auf die physikalische Intensität oder Leuchtkraft der Farbe und ist unabhängig von der tatsächlichen Farbabtönung und Sättigung.

Mathematisch gesehen sind beide Modelle bei der Beschreibung von verschiedenen Farben schwer zu behandeln. Oft ist das beste Mittel zur Messung einer unbekannten Farbe der Vergleich mit einer Mustersammlung, wie die Farbtafeln, die man beim Kauf einer Lackdose verwendet.

Eine objektivere Methode beim Farbvergleich ist die Klassifizierung der Wahrnehmung (normalerweise sicht-

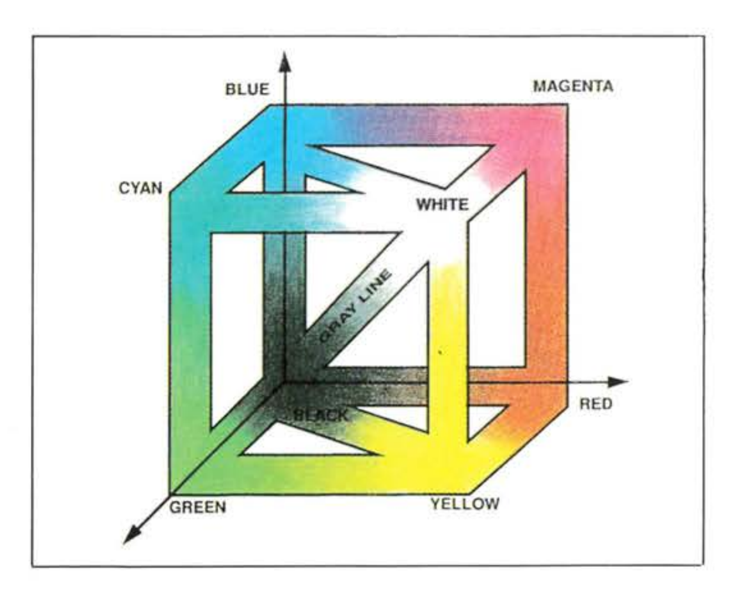


Bild 2: *Ein RGB-Farbwürfel (hier als Drahtmodell dargestellt, damit Farben von allen Seiten gezeigt werden können).*

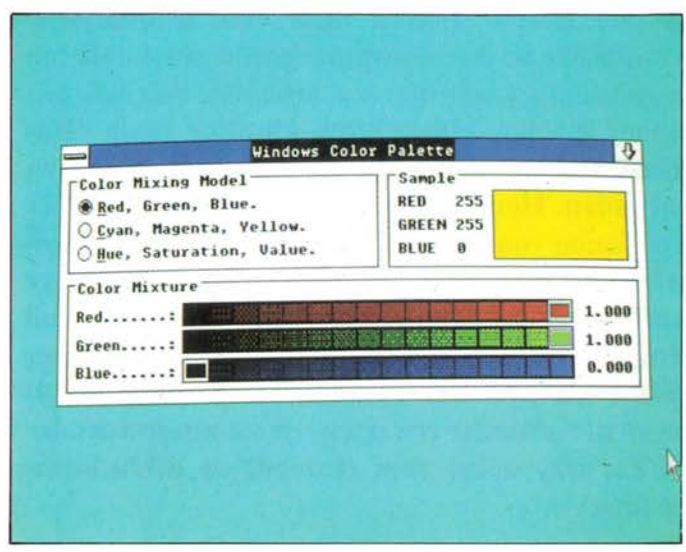


Bild 3: *PALETTE wird zum Mischen von Grundfarben verwendet.*

bares Licht) nach Hauptwellenlänge, Reinheit und Leuchtstärke. Die daraus resultierende spektrale Energieverteilung ergibt dann die »Farbe«. Der Ausdruck Hauptwellenlänge ergibt sich aus der Tatsache, daß unser Auge beim Betrachten eines Objekts auf eine spezifische Wellenlänge stärker als auf andere reagiert. Diese bestimmt die Farbe, die wir sehen. Reinheit bezieht sich auf die mathematische Kombinierung von farblichem und weißem Licht. Eine vollständig reine Farbe enthält kein weißes Licht und wird daher oft als 100 Prozent gesättigt bezeichnet. Leuchtstärke bezieht sich auf die Gesamtenergie einer bestimmten Farbe. Dies beinhaltet sowohl die Reinheit als auch die Hauptwellenlänge des zu betrachtenden Lichts - je mehr Energie, desto sichtbarer die bestimmte Farbe.

Farb-Mischmodelle

Von der physiologischen Seite her wird theoretisiert, daß die Retina des menschlichen Auges drei Arten von Zäpfchen enthält, empfindlich für Rot, Grün und Blau. Diese Hypothese wurde, trotz anderer neuer Sehtheorien, auf das Gebiet der Computergrafik angewendet. Die meisten Farbmonitore verwenden rotes, grünes und blaues Phosphor beim Versuch, das menschliche Auge zu simulieren.

Die Anwendung der Rot-Grün-Blau-Theorie bei Farbmonitoren und anderen Hardware-Systemen hat beträchtliches Interesse für die Entwicklung von Modellen ausgelöst, die sich diese praktische Spezifizierung von Farbe zunutze machen. Die daraus resultierenden Modelle beeinflussen unsere Annahme, wie diese Systeme Farbe verwenden. Obgleich sich bestimmte Methoden in einem Fall bewähren, in anderen nicht, können sie alle als gleichwertig angesehen werden, da dieselben Resultate damit erzielt werden können (mit verschiedenen Schwierigkeitgraden).

RGB-Mischmodelle

Das Rot-Grün-Blau-Mischmodell (RGB) ist hardwareorientiert und eine einfache Erweiterung der physikalischen Eigenschaften von Farbmonitoren (siehe *Bild 2*). In diesem Modell sind die drei Hauptfarben in einem kartesischen Koordinatensystem kombiniert. Das Ergebnis dieser Mischung kann sich von den verwendeten Primärfarben stark unterscheiden. Zum Beispiel ist Schwarz das Fehlen jeglicher Farbe (dargestellt als der RGB-Dreifachwert 0-0-0), und die 100-prozentige Mischung aller Farben ergibt Weiß (1-1-1). Die Mischung zweier Primärfarben erzeugt eine sekundäre Farbmischung. Dies sieht man bei der Erzeugung von Gelb (1-1-0) durch Mischen von reinem Rot (1-0-0) und reinem Grün (0-1-0).

Man beachte, daß im RGB-Modell jede Hauptfarbe im Bereich (0,1) liegt. In den meisten tatsächlichen Anwendungen, die auf diesem Modell beruhen, wird eine diskrete Umsetzung aus diesem Bereich durchgeführt. Microsoft Windows beispielsweise unterstützt über das Graphic Device Interface (GDI) nur Farben im Integerbereich (0,255).

PALETTE, ein Windows-Farbmischprogramm, das im folgenden Artikel beschrieben wird, unterstützt das RGB-Mischmodell, indem es die ausgewählte Farbe in ein adäquates, von GDI verwendetes Bildschirm-Format umwandelt. Dies geschieht mit den beiden folgenden einfachen Formeln:

```
crtValue = rgbValue * 255
rgbValue = crtValue / 255
```

Mit PALETTE kann man einfach mit verschiedenen Mengen der Hauptfarben experimentieren. In *Bild 3* sieht

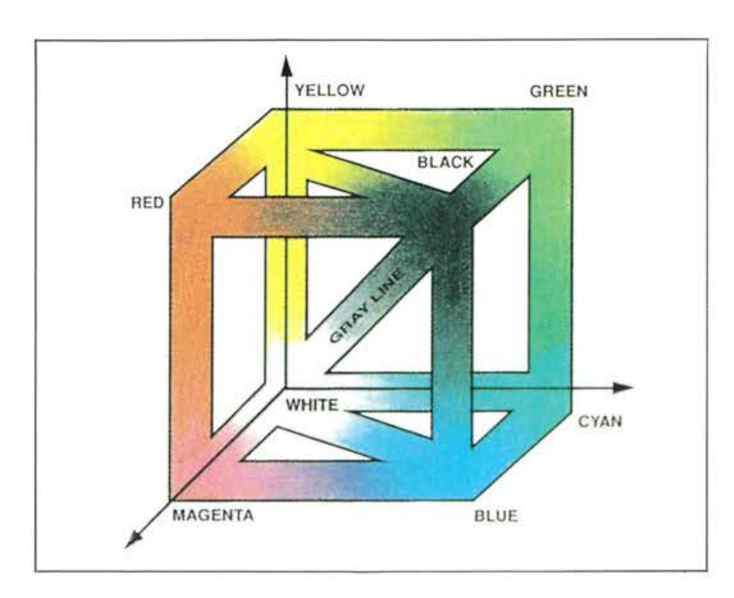


Bild 4: *Ein CMY-Farbwürfel (hier als Drahtmodell dargestellt, damit Farben von allen Seiten gezeigt werden können).*

man, daß die Kombination von reinem Rot und reinem Grün reines Gelb ergibt, immer vorausgesetzt, Sie besitzen einen Farbbildschirm. Wenn Sie weitersuchen, finden Sie vielleicht noch einige andere reine Farben, die nicht in den Ecken des RGB-Farbwürfels auftreten. Versucht man dasselbe Experiment mit anderen Bildschirmsystemen, erhält man wahrscheinlich jeweils verschiedene Ergebnisse.

Wenn Sie einen Monochrom-Monitor verwenden, werden die Farben hoffentlich als jeweils verschiedene Schwarz-Weiß-Mischungen erscheinen. In einem Farbsystem hängt die Anzahl der möglichen reinen Farben von den jeweiligen Eigenschaften der Hardware und des verwendeten GDI-Bildschirmtreibers ab. Einige Treiber sind sehr gut bei der Annäherung von Farben, die zwischen Hauptfarben liegen. Andere hingegen verwenden die nächstliegende Hauptfarbe.

Beim Experimentieren werden Sie feststellen, daß die Rot-, Grün- und Blau-Spektren, für sich selber wiederum nur Muster aus dem RGB-Würfel darstellen. Diese Muster wurden durch einen einfachen Algorithmus erzeugt, der die Ecken des RGB-Farbwürfels umläuft:

```
for ( i=0, i<16, i++)
{
    rgbValue = i / 15.00;
    crtValue = rgValue * 255;
}
```

Sie können Ihre eigene Mischtabelle durch Einsetzen eines anderen Algorithmus erzeugen.

Das CMY-Mischmodell

Das Cyan-Magenta-Gelb-Mischmodell (CMY), das Hauptfarben subtrahiert, anstelle zu addieren, ist eine Variation des RGB-Modells. Anders gesagt, der gewünschte Effekt wird dadurch erzielt, daß Farben vom weißen Licht abgezogen werden. Sind alle Farben vertreten, ergibt sich Weiß, ist keine vorhanden, Schwarz. Es ist sehr wichtig, das CMY-Modell zu verstehen, da es von vielen Hardcopy-Ausgabegeräten verwendet wird. Die Idee, Farbe oder Farbstoff auf weißes Papier aufzutragen, wird bei Plottern, Tintenstrahl- und Laserdruckern verwendet.

Mathematisch kann das CMY-Modell durch Einsatz eines Kartesischen Farbwürfels erklärt werden (siehe *Bild 4*). Das resultierende Sub-Koordinatensystem ist mit dem RGB-Würfel identisch, mit der Ausnahme, daß Weiß anstelle von Schwarz im Ursprung liegt. Beim Anblick des CMY-Würfels erkennt man, daß eine der Diagonalen vom schwarzen zum weißen Eckpunkt verläuft. Farb-Triple entlang dieser Linie ergeben verschiedene Grau-Mischungen. Mit PALETTE kann man sowohl im RGB- als auch im CMY-Modell Grauwerte dadurch erzielen, daß für jede der Hauptfarben der gleiche Intensitätswert verwendet wird.

Mit PALETTE werden Sie noch etwas anderes über die RGB- und CMY-Mischmodelle feststellen: Ein Grauwert wird immer dann erzielt, wenn eine Farbmischung alle Hauptfarben beinhaltet. Angenommen, sie haben beispielsweise eine Mischung aus gleicher Menge Cyan und Magenta, die einen Blauton ergibt. Fügen Sie nun fortlaufend Gelb dazu (bis zum gleichen Grad wie die anderen beiden Farben) verblaßt das Blau immer mehr zu Grau. Leider ist dies schwieriger, wenn man mit einer Einfärbung beginnt, die ungleiche Mengen von Cyan und Magenta enthält. Glücklicherweise stehen das RGB- und CMY-Modell in einem einfachen Verhältnis zueinander:

```
Cyan = 1-Rot
Magenta = 1-Grün
Gelb = 1-Blau
```

Mit dieser Relation kann man einfach eine Transformation erzeugen, die jeden beliebigen CMY-Wert in einen RGB-Wert umwandelt und umgekehrt.

Das YIQ-Modell

Das YIQ-Modell kann, ebenso wie das RGB- und CMY-Modell, durch einen Kartesischen Koordinatenwürfel beschrieben werden. Dieses Modell ist insofern interessant, als es eine Recodierung des RGB-Modells darstellt, optimiert für Übertragungseffektivität und Abwärts-Kompatibilität mit monochromen Subsystemen, wie beispielsweise Schwarz-Weiß-Fernsehern.

Die Y-Komponente dieses Modells stellt eine Grundfarbe dar, deren spektrale Energieverteilung der Helligkeits-Kurve entspricht, wobei die Y-Komponente äquivalent zur definierten Farbe ist. Dadurch wird ein wichtiges Problem des Fernsehens gelöst, nämlich der Empfang desselben Signals von Farb-, als auch Schwarz-Weiß-Geräten. Im RGB-Modell können zwei verschiedene Schattierungen auf

```
Y = (0,30*Rot)+(0,59*Grün)+(0,11*Blau)
I = (0,60*Rot)+(-0,28*Grün)+(-0,32*Blau)
Q = (0,21*Rot)+(-0,52*Grün)+(0,31*Blau)
```

Tabelle 1: *Diese Formel berechnet das YIQ-Äquivalent zu RGB-Werten.*

einem Farbfernseher die gleiche Helligkeit bei einem Schwarz-Weiß-Gerät ergeben. Im YIQ-Modell ist dieses Problem durch Umsetzung der unterschiedlichen Farben auf entsprechend unterschiedliche Intensitätsgrade gelöst.

Obwohl in PALETTE das YIQ-Mischmodell nicht implementiert ist (das ist einfach zu bewerkstelligen, falls Sie dies einmal wollen) kann man das entsprechende YIQ-Äquivalent zu einem RGB-Farbwert sehr leicht mittels der Formel aus *Tabelle 1* berechnen.

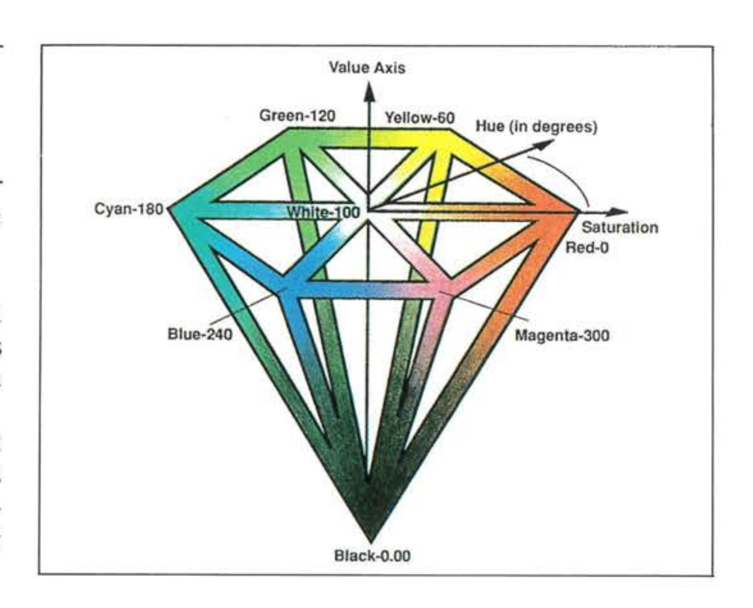


Bild 5: *Ein HSV-Hexakegel (hier als Drahtmodell dargestellt, damit Farben von allen Seiten gezeigt werden können).*

HSV-Mischmodell

Das HSV-Mischmodell ist im Gegensatz zu den drei vorherigen hardwareorientierten Modellen ein Anwendermodell, basierend auf der Auffassung der Künstler von Farbgebung, Sättigung und Wert. Dieses Modell hat einige deutliche Vorteile gegenüber den anderen.

Topologisch betrachtet ist das HSV-Modell ein Hexakegel, ein auf den Kopf gestellter, sechsseitiger Kegel (*Bild 5*). Der Fuß dieses Kegels entspricht V=0 und stellt Schwarz dar. Das obere Ende des Kegels stellt Weiß mit V=1 dar. Jeder der Scheitelpunkte stellt eine Primärfarbe oder sekundäre Farbe dar und entspricht einem H-Wert. Rot beispielweise entspricht einer Farbabtönung von 0 Grad, Gelb 60 Grad und so weiter. Die verbleibende Komponente Sättigung ist ein Vektor, der horizontal vom Ursprung ausgeht. Solange S den Wert 0 beibehält, ist die definierte Farbe immer ein Grauwert, der von Schwarz bis Weiß reichen kann. Ist S größer als Null, liegt ein bestimmter Farbton vor. Dieser Farbton wird immer gesättigter bis S=1. Jede Farbe, die man bei S=1 und V=1 erhält, entspricht einer Pigmentfarbe, wie sie Künstler benutzen.

Läßt man die Sättigung konstant, angenommen gleich 1, erhält man alle primären und sekundären Einfärbungen, indem man die Farbabtönungen von 0 bis 360 Grad durchläuft. Ist eine bestimmte Farbe gefunden, kann man diese leicht durch fortwährendes Reduzieren der Sättigung auf Null, zu Weiß, ausbleichen. Ebenso kann jede Einfärbung durch Reduzieren von V zu Schwarz abgeblendet werden.

Beim Experimentieren mit PALETTE und dem HSV-Modell sieht man bald, wie intuitiv diese Methode ist. So sind einige der relativ einfach erhaltenen Farbmischungen nur schwer mit einem der anderen Modelle zu erreichen.

Das mitgelieferte voreingestellte Spektrum für dieses Modell ergab sich aus einer Auswahl verschiedener Pfade durch den Hexakegel.

Das Standard-Farbabtönungs-Spektrum wurde berechnet, indem S=1 und V=1 konstant gehalten wurden und H die Werte von 0 bis 360 Grad durchlief. Man beachte dabei, daß 0 und 360 äquivalent sind - hat man einmal die vollen 360 Grad durchlaufen, kehrt man wieder zum Startpunkt zurück. Der höchste Wert, den man mit PALETTE auf einer linearen Skala erreichen kann, ist 340. Der nächste Schritt führt zurück auf 0 (bzw. 360). Das Sättigungs-Spektrum wurde berechnet mit H=240 (Blau) und V=1. In diesem Fall werden mehrere S-Werte verwendet, die von 0 bis 1 reichen. Das voreingestellte Wertespektrum schließlich wurde bestimmt, indem V schrittweise von 0 bis 1 erhöht wird, bei konstanten S=0 und H=0.

Ein interessanter Aspekt der HSV-Methode ist seine Beziehung zum RGB-Modell. Betrachtet man den RGB-Farbwürfel entlang seiner Hauptdiagonalen, wirkt er wie ein Schnitt durch den Hexakegel bei konstantem V. Jede Schnittebene bei konstantem V entspricht dabei einem Sub-Würfel im RGB-Würfel (*Bild 6*).

Windows PALETTE

Im nachfolgenden Artikel werden Sie bemerken, daß am Ende der Module PALETTE.C mehrere Farbkonvertierungsroutinen stehen, die PALETTE zum Arbeiten mit dem jeweiligen Mischmodell verwendet. Sobald Sie eine neue Farbe definieren, konvertiert das Programm diese automatisch in eine äquivalente Darstellung in einem der anderen Modelle. Ebenso sucht PALETTE, wenn Sie ein neues Mischmodell auswählen (mit den Optionsfeldern), darin nach dem besten Gegenstück der aktuellen Farbe.

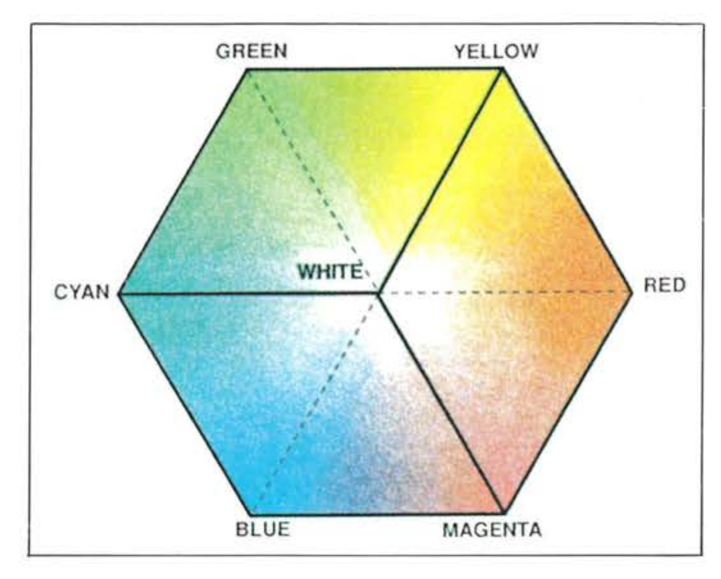


Bild 6: *Der RGB Sub-Würfel.*

Routine	Bedeutung
RGBtoCRT()	konvertiert vom RGB- ins CRT-Modell
RGBtoCMY()	konvertiert vom RGB- ins CMY-Modell
RGBtoHSV()	konvertiert vom RGB- ins HSV-Modell
CRTtoRGB()	konvertiert vom CRT- ins RGB-Modell
CMYtoRGB()	konvertiert vom CMY- ins RGB-Modell
HSVtoRGB()	konvertiert vom HSV- ins RGB-Modell

Tabelle 2: *Die dargestellten Routinen werden von PALETTE benutzt, um die Farbwerte von einer Mischmethode zur anderen zu konvertieren.*

Dieser mathematische Interpolation- und Suchvorgang ist etwas ungenau. Er gibt Ihnen jedoch trotzdem eine gute Vorstellung davon, wie man eine ähnliche Farbe in einem anderen Modell erzeugt. Obgleich in den meisten Fällen das jeweilige Gegenstück genau gleich ist, kann doch die quantenmäßige Natur der Spektralsteuerung zu einigen Inkonsistenzen führen.

Die Routinen, die diese Farbkonvertierungen in PALETTE.C durchführen, sind in *Tabelle 2* aufgelistet. In diesem Artikel haben wir alle entsprechenden Algorithmen aufgezeigt, ausgenommen jenen für das HSV-Modell. Dieser ist etwas komplizierter (mathematisch eine Art topologische Windung) und wird besser einer anderen Erläuterung überlassen. Sollten Sie näher daran interessiert sein, ist einige Literatur über Computergrafik erhältlich, die dieses Modell behandelt und den Konvertierungs-Algorithmus von RGB-in-HSV enthält.

PALETTE bietet neben einem interessanten Programm, das den Nutzen einer abgestimmten Steuerung durch Dialogboxen zeigt, auch einen Einblick in die Mischmodelle und deren gerätespezifische Abhängigkeiten.

Dieser Artikel hat Ihnen, so hoffen wir, ein besseres Verständnis dieser Grundlagen geliefert und einige der Unsicherheiten mit Windows GDI beseitigt. Darüber hinaus werden Ihnen vielleicht die Farbkonvertierungs-Routinen bei Ihrer nächsten Entwicklung einer Windows-Anwendung von Nutzen sein.

Zahlreiche gute Literatur ist erhältlich, mit eingehenden Abhandlungen über Farben, Farbmischmodelle und interaktive Computergrafik im allgemeinen. Eines der hilfsreichsten Bücher davon ist *Fundamentals of Interactive Computer Graphics* von J.D. Foley und A. Van Dam, veröffentlicht bei Addison Wesley).

Kevin P. Welch

Inserentenverzeichnis

Windows-Anwendungen intelligent erweitern:

Selbstdefinierte Dialog-Steuerungen

Microsoft Windows scheint, so wie auch andere Grafik-Umgebungen, an einer Einschränkung zu leiden, über die sich Programmierer fortlaufend beschweren. Sie sind beschränkt auf eine vorgegebene Anzahl von Tools, die den Rechner standardisieren. Es gibt jedoch, in der Tiefe des Windows Software Development Kit (SDK) vergraben, Anhaltspunkte, das dem nicht so sein muß. Die kooperative Natur von Windows bietet einige Mechanismen, die zur Erzeugung von neuen und ganz unterschiedlichen Software-Objekten herangezogen werden können.

Eines der interessantesten davon ist die Möglichkeit, eigene Dialogbox-Steuerungen herzustellen. Dadurch kann man Windows funktionell erweitern, Anwendungen visuell ansprechender gestalten und die Anwenderschnittstelle erweitern. Es lassen sich beispielsweise Steuerungen zur Darstellung von Schaltern, Meßzeigern, Linealen und anderen nützlichen Einrichtungen definieren. Wir werden uns den Arbeitsvorgang anhand einer bestimmten Steuerung namens Spectrum anschauen und dessen Einsatz in einem Farbmischprogramm namens Palette. Im Artikel auf den vorangegangenen Seiten haben wir theoretisch verschiedene Mischmodelle erforscht. Palette kann zur Veranschaulichung von drei dieser Darstellungsarten herangezogen werden.

Grundlagen von Steuerungen

Fast alles Wichtige in der Windows-Umgebung ist eine Art von Fenster. Standard-Dialogbox-Steuerungen sind ebenfalls, wie Sie sicher erraten haben, vordefinierte Fenster, die in Ihrem gesamten Anwendungsprogramm verwendet werden können.

Diese Steuerungen gleichen den Fenstern sehr, die auch in Ihren eigenen Programmen registriert oder erzeugt werden. Steuerungen sind jedoch, im Gegensatz zu den meisten von Ihnen erzeugten Fenstern, notwendigerweise objektorientiert und bestehen vollständig aus reentrantem Code. Dies hat zur Folge, daß eine Steuerung weder auf statische Daten zugreifen sollte, noch von der Tatsache abhängig sein soll, ob gerade nur ein Ausführungs-Thread vorhanden ist. Um eine korrekte Bearbeitung durch den Dialogbox-Manager zu gewährleisten, müssen auch alle Tastatur- und Mauseingaben auf konsistente Art behandelt werden.

Obwohl die Detailarbeiten bei der Implementierung einer Steuerung ziemlich kompliziert werden können, sind die Grundschritte sehr einfach.

Bestimmen eines Klassennamens: Jeder Steuerung muß ein eindeutiger Name zugewiesen werden. Der gleiche Name wird auch in Ihrer RC-Datei verwendet, sobald die Steuerung in einer Dialogbox verwendet werden soll.



Bild 1: *Das Windows-Programm PALETTE.*

Dabei müssen Sie darauf achten, keinen von Windows vordefinierten Klassennamen zu wählen. Die Resultate können ziemlich spektakulär ausfallen, wenn ein solcher Name versehentlich vergeben wird.

Bestimmen aller Instanzenvariablen. Instanzenvariablen sind private statische Daten, die mit den jeweiligen Instanzen verbunden sind. Diese Daten müssen dynamisch zugewiesen und im Zugriff für alle ausgeführten Threads stehen. Obgleich Sie Ihre eigenen Strukturen für Instanzenvariablen entwickeln können, ist es trotzdem günstig, globale oder lokale Atome und Eigentumslisten zu verwenden, oder vielleicht extra eine Anzahl von Bytes zu definieren, die mit jeder Instanz der Fensterklasse-Datenstruktur in Zusammenhang steht.

Definition Ihrer Steuerungs-Fensterfunktion: Jeder Steuerung muß eine Fensterfunktion zugewiesen werden, die alle entsprechenden Meldungen bearbeitet. Da Ihre Steuerung vermutlich vom Dialogbox-Manager überwacht wird, sollten Sie die Implementierung sowohl einer Tastatur als auch einer Mausschnittstelle in Betracht ziehen. Zumindest sollte Ihre Fensterfunktion folgende Meldungen berücksichtigen:

- `WM_CREATE`: Wenn Sie diese Meldung erhalten, sollten Sie jede Instanzenvariable initialisieren und alle anderen notwendigen Initialisierungsschritte vornehmen.
- `WM_PAINT`: Nach Empfang dieser Meldung sollten Sie jede angeforderte Anzeige im Arbeitsbereich Ihres Fensters durchführen. Sie sollten aufpassen, daß Sie die Ergebnisse Ihrer Anzeigeoperation für verschiedene Arten von Bildschirmeinheiten vorsehen, speziell jene mit stark abweichendem Pixel/Aspekt-Verhältnis. Dies ist wahrscheinlich der wichtigste und zeitaufwendigste Teil der gesamten Steuerungs-Entwicklung.
- `WM_DESTROY`: Bei Erhalt dieser Meldung sollten Sie jeden Speicherbereich, der von der Steuerungsinstanz belegt ist, freigeben und vor der Vernichtung des Fensters alle notwendigen Löschoperationen durchführen.

DLGC-WANTALLKEYS fängt alle Tastatureingaben ab
DLGC-WANTALLARROWS nur Cursor-/Richtungstasten
DLGC-WANTALLCHARS nur WM_CHAR-Meldungen
DLGC-WANTALLMESSAGE alle Meldungen
DLGC-WANTALLTAB Tabulator-Tasten

Tabelle 1: *Return-Codes für WM_GETDLGCODE*

Falls Sie Ihre Steuerung für Eingaben vorsehen (sowohl von der Tastatur als auch mit der Maus), beachten Sie, daß eine oder mehrere der folgenden Meldungen bearbeitet werden müssen:

- WM_GETDLGCODE: Ihre Antwort auf diese Meldung entscheidet, wie der Dialogbox-Manager Ihre Steuerung weiterverarbeitet. Durch Rückmeldung eines Return-Codes aus *Tabelle 1* bewirken Sie eine bestimmte Art der Eingabe. Die Bearbeitung der Daten wird dabei von Ihnen vorgenommen.
- WM_SETFOCUS: Bei Empfang dieser Meldung sollten Sie eine Schreibmarke erzeugen und anzeigen, die dem Anwender mitteilt, daß Sie im Besitz des Eingabefokus sind.
- WM_KILLFOCUS: Diese Meldung wird empfangen, sobald ein anderes Fenster den Eingabefokus erhält. Haben Sie zu dieser Zeit eine Schreibmarke definiert, sollten Sie diese zerstören.
- WM_KEYDOWN: Diese Meldung wird erhalten, sobald eine Taste gedrückt oder niedergehalten wird. Durch Ausführung der virtuellen Tastencodes können Sie jede nötige Operation durchführen. Stellen Sie auch sicher, daß dem übergeordneten Fenster jede Aktion gemeldet wird, die zu einem neuen Systemzustand führt, wie das Drücken einer Schaltfläche oder die Auswahl eines Objekts aus einer Objektliste.
- WM_LBUTTONDOWN: Diese Meldung wird empfangen, sobald der Anwender in dem Steuerungsfenster die linke Maustaste drückt. In den meisten Fällen sollten Sie den Systemfokus übernehmen, die Mausbewegungen abfangen und Ihre Schreibmarke abbilden. Wenn durch die Maus eine neue Auswahl getroffen wird, ist auch das übergeordnete Fenster von der Veränderung zu benachrichtigen.
- WM_MOUSEMOVE: Diese Meldung kann man in der Regel ignorieren, obwohl eine Aktion manchmal sinnvoll ist, wenn Sie in Zusammenhang mit einer Maus-Zieh-Operation auftritt.
- WM_LBUTTONUP: Diese Meldung wird empfangen, sobald der Anwender den linken Mausknopf freigibt. Wenn Sie Mausbewegungen abfangen, sollten Sie darauf achten, daß ab diesem Zeitpunkt die Abfangkontrolle wieder von Ihnen freigegeben wird. Achten Sie darauf, daß die Meldungen WM_LBUTTONDOWN und WM_LBUTTONUP immer auftreten können, und Sie darum sicherstellen müssen, daß diese Meldungen ohne sinnvollen Zusammenhang ignoriert werden.

Registrierung der Steuerungsklasse: Bevor Sie eine Steuerung in einer Anwendung verwenden können, müssen Sie die Steuerungsfensterklasse registrieren. Jeder Versuch, eine Steuerung zu verwenden, bevor sie registriert wurde, führt zu einem schweren Systemfehler und kann sogar die Ausführung Ihrer Anwendung gänzlich verhindern. In der Praxis ist es eine gute Idee, eine eigene spezielle Funktion, als Teil des gesamten Steuerungs-Pakets, mit dieser Aufgabe zu betrauen.
Exportieren Sie die Steuerungs-Fensterfunktion: Sie müssen diese Fensterfunktion, wie jede andere auch, vor Gebrauch exportieren. Selbst erfahrene Windows-Programmierer vergessen dies oft.
Definition von zugehörigen Dienstfunktionen: Selbst die Entwicklung einer einfachen Steuerung erfordert oft die Definition einer Anzahl von zugehörigen Dienstfunktionen, die der Applikation ermöglichen, verschiedene steuerungsabhängige Parameter zu definieren oder zu übernehmen. Obgleich eine sorgfältig definierte Meldungsstruktur manchmal ausreicht, ist es in komplizierteren Situationen besser, eine eigene spezielle Funktion zu entwickeln, die jede erforderliche Aktion ausführt. Dies erlaubt der Steuerung ein höheres Maß an Eigenständigkeit und fördert die Entwicklung von selbständigen Software-Objekten.
Schließen Sie die Steuerung in die gewünschten RC-Dateien ein: Im allgemeinen sollten Sie den Dialogbox-Editor (Teil des Windows-SDK) für die Entwicklung der Schablonen verwenden, die in die RC-Datei übernommen werden sollen. Leider unterstützt der Dialogbox-Editor anwenderdefinierte Steuerungen nicht. Ich fand es hilfreich, die Größe und Position einer Steuerung mit einem statischen Rahmen zu definieren. Danach editiere ich den resultierenden DLG-Dateiinhalt und ersetze den Rahmen mit der passenden Steuerung. Beachten Sie dabei, daß der Dialogbox-Editor keine RES-Dateien bearbeiten kann, die Verweise auf anwenderdefinierten Steuerungen enthalten.

Spectrum

Der beste Weg, die Definition und Implementierung von anwenderdefinierten Steuerungen zu erklären, führt über ein Beispiel. Spectrum ist deshalb als Studienobjekt besonders interessant, weil es eine gut entwickelte Anwenderschnittstelle besitzt, die sowohl lokale als auch globale Instanzendaten ausnützt.

Die Spectrum-Steuerung war das Ergebnis meiner Frustration über den Mangel an systemdefinierten Mechanismen bei der Farbauswahl. Zu dieser Zeit mußte der Anwender, wenn er mit von mir entwickelten Applikationen

arbeitete, eine oder mehrere Farben aus einer Farbpalette auswählen. Nach einigen Versuchen mit Scrollbalken und Objektlisten entschied ich, daß es günstiger wäre, eine Serie von Steuerungen für den generellen Gebrauch zu entwerfen, um solche oder andere Probleme bei Anwenderschnittstellen zu lösen. Das Ergebnis dieser Entwicklung war der Entwurf einer Bibliothek von Steuerungen, welche von Meßreglern, Schaltern und Schaltflächen bis zu Schiebereglern und anderen Selektionswerkzeugen reicht. Die Spectrum-Steuerung ist eine vereinfachte Version eines für diese Bibliothek erzeugten Tools.

Mit Spectrum können Sie eine Folge von Farben definieren, aus welcher der Anwender seine Auswahl treffen kann. Jede Farbe, die von Spectrum verwaltet wird, ist mit einem numerischen Wert verbunden, der dem übergeordneten Fenster übergeben wird, sobald eine neue Auswahl getroffen wird. Dies kann ein Zeichen, Integer, Long-Integer oder Floating Point sein. In bestimmten Situationen kann dieser numerische Wert sogar mit dem abgebildeten Rot-Grün-Blau-Farbwert (RGB) übereinstimmen.

Die derzeit aktuelle Farbe wird durch eine kleine rechteckige Markierung, die diese umschließt, gekennzeichnet. Falls die Spectrum-Steuerung den Systemfokus besitzt, ist die System-Schreibmarke ebenfalls innerhalb des Rechtecks abgebildet. Wenn die Schreibmarke sichtbar ist, kann die Auswahlmarke mit den Cursortasten oder durch die Maus bewegt werden.

Vor Anwendung der Spectrum-Steuerung in einer Dialogbox muß diese durch die Funktion `RegisterSpectrum` registriert werden. *Listing 1* zeigt einen Teil der Spectrum-Funktionen. (Die vollständigen Quellcode-Listings und Befehlsdateien für Spectrum und Palette finden Sie auf der gesondert erhältlichen Diskette.) Der einzige ungewöhnliche Aspekt dieser Funktion ist der Gebrauch des Feldes `cbWndExtra` in der Datenstruktur `WndClass`. Der angegebene Wert definiert die Anzahl der zusätzlichen Bytes, die für jede Instanz des Spectrum-Fensters zugewiesen werden müssen. Auf diese Bytes kann dann mit den Funktionen `SetWindowWord` und `GetWindowWord` zugegriffen werden. Die verschiedenen Definitionen dafür finden Sie am Anfang von SPECTRUM.C. Vergleicht man die Offsets mit den vorgegebenen Offsets für `GetWindowWord` in WINDOWS.H, erkennt man, daß die in Windows.H negativ sind. Dies bedeutet, daß die unter `cbWndExtra` zugewiesenen Bytes immer positiv sind und bei Offset 0 beginnen.

Die Funktion `SetSpectrum` ist eine kleine Dienstfunktion, die Ihnen die Definiton eines ausgewählten Farbmusters von der derzeit aktuellen Palette erlaubt. Der Selektor-Index wird dabei als Instanzenvariable mit einem vorgegebenen Fenster-Offset abgespeichert. Analog zu `SetSpectrum` ist auch `GetSpectrum` eine Funktion, mit der man einen Index zu dem aktuellen Farbmuster erlangen kann. Trotz der Einfachheit dieser Funktion, ist sie sehr nützlich, weil dadurch die Applikation von den internen Details der Spectrum-Steuerung ausgeschlossen wird.

```
/*
 * WINDOWS SPECTRUM CONTROL
 *
 * LANGUAGE : Microsoft C 5.0
 * TOOLKIT  : Windows 2.03 SDK
 * MODEL    : Small or Medium
 * STATUS   : Operational
 *
 * 03/20/88 - Kevin P. Welch - initial creation.
 *
 */

#include <windows.h>
#include "spectrum.h"

.
.
.

/*
 * RegisterSpectrum( hAppInstance ) : BOOL
 *
 *    hAppInstance      Applikation-Instanzen-Handle
 *
 * Diese Funktion dient zur Definition und Registrierung
 * der Spectrum-Fensterklasse. Sie sollte nur EINMAL
 * aufgerufen werden (Typischerweise in der Initialisierungsphase
 * der Host-Applikation) innerhalb eines Programms.
 *
 * Ein Wert TRUE wird zurückgeliefert, falls die Registrierung
 * erfolgreich war.
 *
 */

BOOL FAR PASCAL RegisterSpectrum( hAppInstance )
   HANDLE      hAppInstance;
{
   /* Lokale Variablen */
   WNDCLASS    WndClass;   /* Fensterklassen-Datenstruktur */
   /* Definiert die Spectrum-Fensterklasse. Es wird nicht geprüft, ob
    * bereits früher eine Registrierung stattfand! Es ist Aufgabe
    * der Host-Applikation, mehrmaliges Registrieren zu verhindern.
    */

   memset( &WndClass, 0, sizeof(WNDCLASS) );

   WndClass.lpszClassName =   (LPSTR)"Spectrum";
   WndClass.hCursor =         LoadCursor( NULL, IDC_ARROW );
   WndClass.lpszMenuName =    (LPSTR)NULL;
   WndClass.style =           CS_HREDRAW|CS_VREDRAW;
   WndClass.lpfnWndProc =     SpectrumWndFn;
   WndClass.hInstance =       hAppInstance;
   WndClass.hIcon =           NULL;
   WndClass.cbWndExtra =      SPECTRUM_EXTRA;
   WndClass.hbrBackground =   (HBRUSH)(COLOR_WINDOW + 1 );

   /* Registriert Spectrum-Fensterklasse & Return */
   return( RegisterClass( (LPWNDCLASS)&WndClass ) );

}

.
.
.

/*
 * SetSpectrumColors( hWnd, pwRange, prgbList ) : BOOL
 *
 *    hWnd           Handle auf Spectrum-Steuerung in Dialogbox
 *    pwRange        Anzahl der Farbeinträge in RGB-Liste
 *    prgbList       Liste der RGB-Farben & Werte für Spectrum
 *
 * Ermöglicht dem Aufrufer, ein neues Spektrum zu definieren,
 * mit zugehörigen Lookup-Werten für die Steuerung. Die
 * aufrufende Routine ist für die Bestimmung der passenden RGB-Farben
 * und Tabellen-Werte verantwortlich. Die gewählte Farbe ist
 * immer auf den ersten Eintrag gesetzt, sobald ein neuer
 * Bereich definiert wird.
 * Der Wert TRUE wird zurückgeliefert, wenn eine neue RGB-Farbliste
 * erfolgreich definiert wurde.
 *
 */

BOOL FAR PASCAL SetSpectrumColors( hWnd, pwRange, prgbList )
   HWND        hWnd;
   WORD *      pwRange;
   LONG *      prgbList;
{
   /* Lokale Variablen */
   WORD        wEntry;           /* temporärer Farbeintrag */
   HANDLE      hrgbList;         /* Handle auf RGB-Liste */
   LONG FAR *  lprgbEntry;       /* Pointer auf RGB-Liste */
   RECT        rectClient;       /* Anwendungs-Bereich Rechteck */

   /* Gibt vorherige Tabelle aus Speicherbereich frei */
   GlobalFree( TABLE );
   hrgbList = GlobalAlloc( GMEM_MOVEABLE, sizeof(LONG)*(*pwRange)*2L);
   if ( hrgbList ) {

      /* Definiert RGB Anfangsfarben & Werte */
      lprgbEntry = (LONG FAR *)GlobalLock( hrgbList );
      for ( wEntry=0; wEntry < *pwRange; wEntry++ ) {
         lprgbEntry[wEntry] = prgbList[wEntry];
         lprgbEntry[(*pwRange)+wEntry] = prgbList[(*pwRange)+wEntry];
      }
      GlobalUnlock( hrgbList );

      /* Holt aktuelle Fenstermaße */
      GetClientRect( hWnd, &rectClient );
```

Listing 1: *Ausschnitte aus dem Programm* `SPECTRUM.C`.

```
      /* Re-definiert Instanzvariablen */
      SET_RANGE( *pwRange );
      SET_TABLE( hrgbList );
      SET_WIDTH( (rectClient.right-rectClient.left)/(*pwRange) );
      SET_HEIGHT( rectClient.bottom-rectClient.top );
      SET_CHOICE( 0 );

      /* Aktualisiert Fenster & benachrichtigt Parent von neuer Auswahl*/
      InvalidateRect( hWnd, NULL, TRUE );
      SendMessage( PARENT, WM_COMMAND, ID, lprgbEntry[RANGE+CHOICE] );

      /* Normales Return */
      return( TRUE );

   } else
      return( FALSE );

}

.
.
.

/*
 * SpectrumWndFn( hWnd, wMsg, wParam, lParam ) : LONG
 *
 *    hWnd             Handle auf Spectrum-Fenster
 *    wMsg             Nummer der Meldung
 *    wParam           Word-Parameter
 *    lParam           Double-Word-Parameter
 *
 * Bearbeitet alle Meldungen, die zum Spectrum-Steuerungsfenster
 * gehören. Achten Sie darauf, wie der Code geschrieben ist,
 * um mögliche Probleme mit der Reentranz zu vermeiden -
 * dies schließt den Gebrauch von zusätzlichen Bytes, die zur
 * Fenster-Datenstruktur gehören, ein.
 *
 * Der zurückgelieferte LONG-Wert ist das konventionelle Ergebnis
 * der Standard-Fensterprozedur oder der internen Verarbeitung
 * einer Nachricht.
 *
 */

LONG FAR PASCAL SpectrumWndFn( hWnd, wMsg, wParam, lParam )
   HWND        hWnd;
   WORD        wMsg;
   WORD        wParam;
   LONG        lParam;
{
   /* Lokale Variablen */
   LONG        lResult;    /* temporäre Ergebnisvariable */

   /* Initialisierung */
   lResult = TRUE;

   /* Bearbeitet Meldungen */
   switch( wMsg )
      {
   case WM_GETDLGCODE : /* Fängt alle Tastenanschläge ab */
      lParam = DLGC_WANTARROWS;
      break;
   case WM_CREATE :     /* Erzeugt Palette-Fenster */

      {
         /* Zeitweilige Variablen */
         HANDLE      hrgbList;         /* Handle auf RGB-Liste */
         LONG FAR *  lprgbEntry;       /* Pointer auf RGB-Liste */

         /* Allokiert Platz für RGB-Farbliste */
         hrgbList = GlobalAlloc( GMEM_MOVEABLE, sizeof(LONG)*16L );
         if ( hrgbList ) {

            /* Definiert RGB Anfangsfarben & Wertliste - Acht
             * Standardfarben werden gewählt mit den entsprechenden
             * Werten für jede Farbe.
             */

            lprgbEntry = (LONG FAR *)GlobalLock( hrgbList );
            lprgbEntry[0] = RGB( 0x00, 0x00, 0x00 );
            lprgbEntry[1] = RGB( 0x00, 0x00, 0xFF );
            lprgbEntry[2] = RGB( 0x00, 0xFF, 0x00 );
            lprgbEntry[3] = RGB( 0xFF, 0x00, 0x00 );
            lprgbEntry[4] = RGB( 0x00, 0xFF, 0xFF );
            lprgbEntry[5] = RGB( 0xFF, 0xFF, 0x00 );
            lprgbEntry[6] = RGB( 0xFF, 0x00, 0xFF );
            lprgbEntry[7] = RGB( 0xFF, 0xFF, 0xFF );
            lprgbEntry[8] = RGB( 0x00, 0x00, 0x00 );
            lprgbEntry[9] = RGB( 0x00, 0x00, 0xFF );
            lprgbEntry[10] = RGB( 0x00, 0xFF, 0x00 );
            lprgbEntry[11] = RGB( 0xFF, 0x00, 0x00 );
            lprgbEntry[12] = RGB( 0x00, 0xFF, 0xFF );
            lprgbEntry[13] = RGB( 0xFF, 0xFF, 0x00 );
            lprgbEntry[14] = RGB( 0xFF, 0x00, 0xFF );
            lprgbEntry[15] = RGB( 0xFF, 0xFF, 0xFF );
            GlobalUnlock( hrgbList );

            /* Definiert Instanzvariablen */
            SET_RANGE( 8 );
            SET_TABLE( hrgbList );
            SET_WIDTH( ((LPCREATESTRUCT)lParam)->cx / 8 );
            SET_HEIGHT( ((LPCREATESTRUCT)lParam)->cy );
            SET_CHOICE( 0 );
            SET_CAPTURE( FALSE );

         } else
            DestroyWindow( hWnd );
      }

      break;
```

Listing 1: *(Fortsetzung)*

```
   case WM_SIZE : /* Bestimmt Größe des Fensters neu */
      /* Definiert Breite und Höhe der Instanzvariablen neu */
      SET_WIDTH( LOWORD(lParam) / 8 );
      SET_HEIGHT( HIWORD(lParam) );

      break;
   case WM_PAINT : /* Zeichnet Steuerungsfenster */

      {
      PAINTSTRUCT    Ps;            /* Zeichnungsstruktur */
      WORD           wEntry;        /* Aktueller Farbeintrag */
      HANDLE         hBrush;        /* Handle auf neuen Pinsel */
      HANDLE         hOldBrush;     /* Handle auf alten Pinsel */
      LONG FAR *     lprgbEntry;    /* Pointer auf RGB-Liste */

         /* Startet Zeichenoperation */
         BeginPaint( hWnd, (LPPAINTSTRUCT)&Ps );

         /* Zeichnet iterativ alle Farbmuster */
         lprgbEntry = (LONG FAR *)GlobalLock( TABLE );
         for ( wEntry=0; wEntry<RANGE; wEntry++ ) {

            /* Erzeugt vollen Pinsel für Muster & wählt aus */
            hBrush = CreateSolidBrush( lprgbEntry[wEntry] );
            hOldBrush = SelectObject( Ps.hdc, hBrush );

            /* Zeichnet Rechteck mittels Pinsel-Füllung */
            Rectangle(
               Ps.hdc,
               wEntry*WIDTH,
               0,
               (wEntry*WIDTH)+WIDTH,
               HEIGHT
            );

            /* Hebt Pinsel-Auswahl auf und löscht diesen */
            SelectObject( Ps.hdc, hOldBrush );
            DeleteObject( hBrush );

         }
         GlobalUnlock( TABLE );

         /* Definiert aktuelle Auswahl & beendet Zeichenoperation */
         DrawSelector( hWnd, Ps.hdc );
         EndPaint( hWnd, (LPPAINTSTRUCT)&Ps );

      }

      break;
   case WM_KEYDOWN : /* Taste gedrückt */

      {
         /* Lokale Variablen */
         HDC            hDC;           /* Kontext-Handle Bildschirm */
         LONG FAR *     lprgbEntry;    /* Pointer auf RGB-Liste */

      /* Holt Anzeige-Kontext & löscht Markierung aktueller Auswahl */
         hDC = GetDC( hWnd );
         DrawSelector( hWnd, hDC );

         /* Bearbeitet virtuelle Tastenanschläge */
         switch( wParam )
            {
         case VK_HOME : /* Home Taste */
            SET_CHOICE( 0 );
            break;
         case VK_LEFT : /* Linke Cursortaste */
            SET_CHOICE( (CHOICE > 0) ? CHOICE-1 : RANGE-1 );
            break;
         case VK_RIGHT : /* Rechte Cursortaste */
         case VK_SPACE : /* Leertaste - Gehe nach rechts */
            SET_CHOICE( (CHOICE < RANGE-1) ? CHOICE+1 : 0 );
            break;
         case VK_END : /* End Taste */
            SET_CHOICE( RANGE-1 );
            break;
         default : /* Andere Tasten */
            lResult = FALSE;
            break;
         }
         /* Markiert neue Auswahl & gibt Anzeige-Kontext frei */
         DrawSelector( hWnd, hDC );
         ReleaseDC( hWnd, hDC );

         /* Bewegt Schreibmarke zu neuer Position */
         SetCaretPos( CARET_XPOS, CARET_YPOS );

         /* Benachrichtigt Parent von neuer Auswahl */
         lprgbEntry = (LONG FAR *)GlobalLock( TABLE );
         SendMessage(PARENT,WM_COMMAND,ID,lprgbEntry[RANGE+CHOICE]);
         GlobalUnlock( TABLE );

      }

      break;
   case WM_SETFOCUS : /* Holt Fokus - Zeigt Schreibmarke */

      /* Erzeugt Schreibmarke & bildet sie ab */
      CreateCaret( hWnd, NULL, CARET_WIDTH, CARET_HEIGHT );
      SetCaretPos( CARET_XPOS, CARET_YPOS );
      ShowCaret( hWnd );

      break;
   case WM_LBUTTONDOWN : /* Linke Taste gedrückt */

      {
         /* Lokale Variablen */
         HDC            hDC;  /* Handle auf Anzeige */
         LONG FAR *     lprgbEntry; /* Pointer auf RGB Liste */
```

Listing 1: *(Fortsetzung)*

```
        /* Hole Anzeige-Kontext */
        hDC = GetDC ( hWnd );

        /* Löscht Markierung alter Auswahl & markiert neue */
        DrawSelector( hWnd, hDC );
        SET_CHOICE( LOWORD(lParam)/WIDTH );
        DrawSelector( hWnd, hDC );

        /* Gibt Anzeige-Kontext frei & bewegt Schreibmarke */
        ReleaseDC( hWnd, hDC );

        /* Fange Fokus ab & bewege Schreibmarke */
        if ( hWnd == SetFocus(hWnd) )
           SetCaretPos( CARET_XPOS, CARET_YPOS );

        /* Benachrichtige Parent von neuer Auswahl */
        lprgbEntry = (LONG FAR *)GlobalLock( TABLE );
        SendMessage(PARENT,WM_COMMAND,ID,lprgbEntry[RANGE+CHOICE]);
        GlobalUnlock( TABLE );

        /* Aktiviere Abfangen */
        SetCapture( hWnd );
        SET_CAPTURE( TRUE );

     }

     break;
  case WM_MOUSEMOVE : /* Maus wird bewegt */

     /* Folge Maus nur, wenn Abfang eingeschaltet */
     if ( CAPTURE ) {

     /* Lokale Variablen */
     HDC             hDC;          /* Handle auf Anzeige */
     WORD            wNewChoice;   /* Neue Auswahl durch Maus */
     LONG FAR *      lprgbEntry;   /* Pointer auf RGB Liste */

        /* Berechne neue Auswahl */
        wNewChoice = ( *((int*)&lParam) <= 0 ) ?
           0 :
           ( LOWORD(lParam)/WIDTH >= RANGE ) ?
              RANGE - 1 :
              LOWORD(lParam) / WIDTH;

        /* Aktualisiere Anzeige, falls verschieden */
        if ( wNewChoice != CHOICE ) {

           /* Holt Anzeige-Kontext */
           hDC = GetDC ( hWnd );

           /* Löscht alte Auswahlmarkierung & markiert neue */
           DrawSelector( hWnd, hDC );
           SET_CHOICE( wNewChoice );
           DrawSelector( hWnd, hDC );

           /* Gibt Anzeige-Kontext frei & bewegt Schreibmarke */
           ReleaseDC( hWnd, hDC );
           SetCaretPos( CARET_XPOS, CARET_YPOS );

           /* Benachrichtige Parent von neuer Auswahl */
           lprgbEntry = (LONG FAR *)GlobalLock( TABLE );
           SendMessage(PARENT,WM_COMMAND,ID,lprgbEntry[RANGE+CHOICE]);
           GlobalUnlock( TABLE );

        }

     }

     break;
  case WM_LBUTTONUP : /* linke Maustaste wird freigelassen */

     /* Gibt Abfang frei */
     if ( CAPTURE ) {
        SET_CAPTURE( FALSE );
        ReleaseCapture();
     }

     break;
  case WM_KILLFOCUS : /* Löscht Fokus - Verdeckt Schreibmarke */
     DestroyCaret();
     break;
  case WM_DESTROY : /* Zerstört Fenster */
     GlobalFree( TABLE );
     break;
  default : /* Vorgegebene WINDOWS Meldungsbearbeitung */
     lResult = DefWindowProc( hWnd, wMsg, wParam, lParam );
     break;
  }

  /* Liefert Endresultate zurück */
  return( lResult );

}
.
.
.
```

Listing 1: *(Ende)*

Wie bereits erwähnt wird bei Erzeugung einer Instanz für das Spectrum-Fenster ein voreingestellter Satz von Farben automatisch erzeugt. Durch die Funktion `SetSpectrumColor` können das vorgegebene Farbset und die damit verbundenen numerische Werte neu eingestellt werden. Dabei wird die Farbauswahl auf die erste Farbe der Reihe zurückgesetzt und das übergeordnete Fenster von der Änderung benachrichtigt. Die Auswahl der verfügbaren Farben besteht aus einer Reihe RGB-Farben, gefolgt von einer Reihe damit verbundener numerischer Werte, die bei der Auswahl einer Farbe zurückgegeben werden.

Durch die Funktion `GetSpectrumColors` können Sie sich die gerade aktuelle Farbe und Wertetabelle der Spectrum-Steuerung beschaffen. Es wird dabei nicht geprüft, ob genügend Platz für die Liste der Werte vorhanden ist. Dies kann leicht hinzugefügt werden, es ist jedoch der Applikation normalerweise die Menge der Werte bekannt, die verwaltet werden sollen.

Die Funktion `SpectrumWndFN` ist vielleicht der schwierigste Teil des gesamten Programmcodes der Spectrum-Steuerung. Obgleich strukturell sehr einfach aufgebaut, ist sie für die enge Zusammenarbeit mit dem Dialogbox-Manager entworfen. Dies zeigt sich in der Art wie auf `WM_GETDLGCODE` geantwortet wird. Der zurückgelieferte Wert `DLGC_WANTARROWS` benachrichtigt den Dialogbox-Manager, daß die Steuerung die Verwaltung der Cursortasten übernehmen will. Dies erlaubt der Steuerung, eine Tastatur-Anwenderschnittstelle für die Farbauswahl aufzubauen.

Beachten Sie, wie der Befehl `WM_SIZE` verarbeitet wird. Unter normalen Umständen wird meistens keine Größenänderung für Steuerungen vorgenommen. Die Bearbeitung dieses Befehls unterstützt jedoch diese Möglichkeit.

In der Funktion `SpectrumWndFN` sehen Sie einige `SendMessage`-Funktionsaufrufe, die bei jeder Veränderung der Auswahl dem übergeordneten Fenster eine Mitteilung senden. Wie in den meisten anderen Steuerungen werden Informationen mit der Meldung `WM_COMMAND` dem übergeordneten Fenster mitgeteilt. Der `wParam`-Teil dieser Meldung entspricht der Unterfenster-ID (dieser Wert wird mit einem dieser negativen Fenster-Offsets ermittelt) und der `lParam`-Teil ist der von der Auswahlfarbe abhängende numerische Wert.

`DrawSelector` ist die letzte Funktion von SPECTRUM.C. Diese Hilfsfunktion ist für die Erzeugung der Auswahlmarke verantwortlich, die das ausgewählte Farbmuster hervorhebt. Der Zeichenmodus `TRANSPARENT` wird mit dem Operator `R2_NOT` verwendet. Dadurch können weitere Aufrufe der Funktion den Originalzustand der Darstellung wirksam wiederherstellen, wodurch Teile von Spectrum während bei Bewegungen der Auswahlmarke nicht neu gezeichnet werden brauchen.

```
/*
 * WINDOWS COLOR PALETTE UTILITY - SOURCE
 *
 * LANGUAGE : Microsoft C 5.0
 * TOOLKIT  : Windows 2.03 SDK
 * MODEL    : Small
 * STATUS   : Operational
 *
 * 03/20/88 - Kevin P. Welch - initial creation.
 *
 */

#include <windows.h>
#include <math.h>
#include "spectrum.h"
#include "palette.h"

.
.
.

/*
 * WinMain( hInst, hPrevInst, lpszCmdLine, wCmdShow ) : int
 *
 *      hInst           Handle der aktuellen Instanz
 *      hPrevInst       Handle auf vorhergehende Instanz (falls vorh.)
 *      lpszCmdLine     Pointer auf Befehlszeile-Argumente
 *      wCmdShow        Initalisierender ShowWindow Befehl
 *
 * Diese Funktion ist der Systemstart der Applikation
 * und definiert die passenden Fensterklassen und bearbeitet
 * alle Meldungen. Der Dialogbox-Verwalter ist für die
 * Bearbeitung des PALETTE-Fensters verantwortlich.
 *
 *
 */

int PASCAL WinMain( hInst, hPrevInst, lpszCmdLine, wCmdShow )
    HANDLE          hInst;
    HANDLE          hPrevInst;
    LPSTR           lpszCmdLine;
    WORD            wCmdShow;
{
    /* Lokale Variablen */
    FARPROC         lpProc;     /* zeitweilige Funktion */

    /* Registriert Fenster wenn erste Instanz */
    if ( hPrevInst || RegisterSpectrum(hInst) ) {

        /* Zeigt PALETTE Dialogbox */
        lpProc = MakeProcInstance( (FARPROC)PaletteDlgFn, hInst );
        DialogBox( hInst, "Palette", NULL, lpProc );
        FreeProcInstance( lpProc );

    }

    /* Beendet Programm */
    return( FALSE );

}
/*
 * PaletteDlgFn( hWnd, wMsg, wParam, lParam ) : BOOL
 *
 *      hWnd            Handle auf PALETTE Fenster
 *      wMsg            Nummer der Meldung
 *      wParam          Word-Parameter
 *      lParam          Double-Word-Parameter
 *
 * Bearbeitet alle Meldungen, die sich auf die PALETTE-
 * Dialogbox beziehen.  Dies umfaßt hauptsächlich
 * die Definition und Abfrage der verschiedenen Farben,
 * die vom Anwender gewählt sind.
 *
 */

BOOL FAR PASCAL PaletteDlgFn( hDlg, wMsg, wParam, lParam )
    HWND            hDlg;
    WORD            wMsg;
    WORD            wParam;
    LONG            lParam;
{
    /* Lokale Variablen */
    BOOL            bResult;        /* Ergebnis der Funktion */

   /* Initialisierung */
   bResult = TRUE;

   /* Bearbeitet Meldungen */
   switch( wMsg )
    {
   case WM_INITDIALOG : /* Initialisiert Dialogbox */

        /* Wählt CRT Mischmodell */
        wModel = ID_RGB;
        SelectMixingModel( hDlg, wModel );

        /* Definiert Sinnbild für Dialogbox */
        SetClassWord(
            hDlg,
            GCW_HICON,
            LoadIcon( INSTANCE, (LPSTR)"PaletteIcon" )
        );

        break;

.
.
.
```

Listing 2: *Auschnitte aus dem Programm* `PALETTE.C.`

```
case WM_COMMAND : /* Fenster-Befehl */

/* Bearbeitet Sub-Meldungen */
switch( wParam )
    {
case ID_RGB : /* RGB Mischmodell */
case ID_CMY : /* CMY Mischmodell */
case ID_HSV : /* HSV Mischmodell */
        SelectMixingModel( hDlg, wParam );
        break;
case ID_SPECTRUM1 : /* 1. Spektrum */

    switch( wModel )
        {
    case ID_RGB :
        rgbColor.fRed = *((float*)&lParam);
        RGBtoCMY( &rgbColor, &cmyColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE1, "%.3f", rgbColor.fRed );
        break;
    case ID_CMY :
        cmyColor.fCyan = *((float*)&lParam);
        CMYtoRGB( &cmyColor, &rgbColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE1, "%.3f", cmyColor.fCyan );
        break;
    case ID_HSV :
        hsvColor.fHue = *((float*)&lParam);
        HSVtoRGB( &hsvColor, &rgbColor );
        RGBtoCMY( &rgbColor, &cmyColor );
            DlgPrintf( hDlg, ID_VALUE1, "%.0f", hsvColor.fHue );
        break;
    }
    InvalidateRect( GetDlgItem(hDlg,ID_SAMPLE), NULL, NULL );

    break;
case ID_SPECTRUM2 : /* 2. Spektrum */

    switch( wModel )
        {
    case ID_RGB :
        rgbColor.fGreen = *((float*)&lParam);
        RGBtoCMY( &rgbColor, &cmyColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE2, "%.3f", rgbColor.fGreen );
        break;
    case ID_CMY :
        cmyColor.fMagenta = *((float*)&lParam);
        CMYtoRGB( &cmyColor, &rgbColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE2, "%.3f", cmyColor.fMagenta );
        break;
    case ID_HSV :
        hsvColor.fSaturation = *((float*)&lParam);
        HSVtoRGB( &hsvColor, &rgbColor );
        RGBtoCMY( &rgbColor, &cmyColor );
            DlgPrintf( hDlg, ID_VALUE2, "%.3f", hsvColor.fSaturation );
        break;
    }
    InvalidateRect( GetDlgItem(hDlg,ID_SAMPLE), NULL, NULL );

    break;
case ID_SPECTRUM3 : /* 3. Spektrum */

    switch( wModel )
        {
    case ID_RGB :
        rgbColor.fBlue = *((float*)&lParam);
        RGBtoCMY( &rgbColor, &cmyColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE3, "%.3f", rgbColor.fBlue );
        break;
    case ID_CMY :
        cmyColor.fYellow = *((float*)&lParam);
        CMYtoRGB( &cmyColor, &rgbColor );
        RGBtoHSV( &rgbColor, &hsvColor );
            DlgPrintf( hDlg, ID_VALUE3, "%.3f", cmyColor.fYellow );
        break;
    case ID_HSV :
        hsvColor.fValue = *((float*)&lParam);
        HSVtoRGB( &hsvColor, &rgbColor );
        RGBtoCMY( &rgbColor, &cmyColor );
            DlgPrintf( hDlg, ID_VALUE3, "%.3f", hsvColor.fValue );
        break;
    }
    InvalidateRect( GetDlgItem(hDlg,ID_SAMPLE), NULL, NULL );

    break;
case ID_SAMPLE : /* Farbmuster Reihe */

        /* Updated Reihe nur, wenn nötig */
        if ( HIWORD(lParam) == BN_PAINT ) {

            /* Lokale Variablen */
            HDC         hDC;
            CRT         crtValue;
            HANDLE      hOldBrush;
            RECT        rectClient;


            /* Zeigt CRT &quivalente numerische Wert */
            RGBtoCRT( &rgbColor, &crtValue );
            DlgPrintf( hDlg, ID_RED, "RED   %u", crtValue.cRed );
            DlgPrintf( hDlg, ID_GREEN, "GREEN %u", crtValue.cGreen );
            DlgPrintf( hDlg, ID_BLUE, "BLUE  %u", crtValue.cBlue );

            /* Zeichnet Farbenreihe */
        hDC = GetDC( LOWORD(lParam) );
        GetClientRect( LOWORD(lParam), &rectClient );
```

Listing 2: *(Fortsetzung)*

```
            /* Erstellt Anzeige-Kontext */
        hOldBrush = SelectObject(
                hDC,
                CreateSolidBrush( *((DWORD*)&crtValue) )
            );

        /* Zeichnet Farbreihe */
        Rectangle(
            hDC,
            rectClient.left,
            rectClient.top,
            rectClient.right,
            rectClient.bottom
        );

            /* Löscht & gibt Anzeige-Kontext frei */
        DeleteObject( SelectObject(hDC,hOldBrush) );
        ReleaseDC( LOWORD(lParam), hDC );

        }

        break;
    default : /* Etwas anderes */
        break;
    }

    break;
    case WM_NCLBUTTONDBLCLK : /* Anwender-fremder Doppel-Klick */
        EndDialog( hDlg, TRUE );
        break;
    default :
    bResult = FALSE;
    break;
    }

    /* Liefert Ergebnis zurück */
    return( bResult );

}

/*
 * SelectMixingModel( hDlg, wNewModel )
 *
 *     hDlg            Handle auf PALETTE-Dialogbox
 *     wNewModel       ID des neuen Mischmodells
 *
 * Wechselt vom angezeigten Farbenmischmodell
 * zu einem neuen. Dabei wird das Farbspektrum
 * geändert und ein Versuch unternommen, ein
 * Ebenbild der derzeit definierten Farbe im
 * neuen Mischmodell zu finden.
 *
 */

void SelectMixingModel( hDlg, wNewModel )
    HANDLE      hDlg;
    WORD            wNewModel;
{
    /* Lokale Variablen */
    WORD            wRange;
    WORD            wEntry;

    /* Definiert neuen Bereich & prüft radio */
    wRange = 16;
    wModel = wNewModel;
    CheckRadioButton( hDlg, ID_RGB, ID_HSV, wModel );

    /* Initialisiert, in Abhängigkeit vom Modell */
    switch( wNewModel )
        {
    case ID_RGB :

        SetDlgItemText( hDlg, ID_TITLE1, "Red.......:" );
        SetDlgItemText( hDlg, ID_TITLE2, "Green.....:" );
        SetDlgItemText( hDlg, ID_TITLE3, "Blue......:" );

        wEntry = MatchColor(rgbRed,rgbColor.fRed);
        SetSpectrumColors( SPECTRUM1, &wRange, rgbRed );
        SetSpectrum( SPECTRUM1, &wEntry  );

        wEntry = MatchColor(rgbGreen,rgbColor.fGreen);
        SetSpectrumColors( SPECTRUM2, &wRange, rgbGreen );
        SetSpectrum( SPECTRUM2, &wEntry );

        wEntry = MatchColor(rgbBlue,rgbColor.fBlue);
        SetSpectrumColors( SPECTRUM3, &wRange, rgbBlue );
        SetSpectrum( SPECTRUM3, &wEntry  );

        break;
    case ID_CMY :

        SetDlgItemText( hDlg, ID_TITLE1, "Cyan......:" );
        SetDlgItemText( hDlg, ID_TITLE2, "Magenta...:" );
        SetDlgItemText( hDlg, ID_TITLE3, "Yellow....:" );

        wEntry = MatchColor(cmyCyan,cmyColor.fCyan);
        SetSpectrumColors( SPECTRUM1, &wRange, cmyCyan );
        SetSpectrum( SPECTRUM1, &wEntry );

        wEntry = MatchColor(cmyMagenta,cmyColor.fMagenta);
        SetSpectrumColors( SPECTRUM2, &wRange, cmyMagenta );
        SetSpectrum( SPECTRUM2, &wEntry );

        wEntry = MatchColor(cmyYellow,cmyColor.fYellow);
        SetSpectrumColors( SPECTRUM3, &wRange, cmyYellow );
        SetSpectrum( SPECTRUM3, &wEntry );

        break;
```

Listing 2: *(Fortsetzung)*

```
    case ID_HSV :

        SetDlgItemText( hDlg, ID_TITLE1, "Hue.......:" );
        SetDlgItemText( hDlg, ID_TITLE2, "Saturation:" );
        SetDlgItemText( hDlg, ID_TITLE3, "Value.....:" );

        wEntry = MatchColor(hsvHue,hsvColor.fHue);
        SetSpectrumColors( SPECTRUM1, &wRange, hsvHue );
        SetSpectrum( SPECTRUM1, &wEntry );

        wEntry = MatchColor(hsvSaturation,hsvColor.fSaturation);
        SetSpectrumColors( SPECTRUM2, &wRange, hsvSaturation );
        SetSpectrum( SPECTRUM2, &wEntry );

        wEntry = MatchColor(hsvValue,hsvColor.fValue);
        SetSpectrumColors( SPECTRUM3, &wRange, hsvValue );
        SetSpectrum( SPECTRUM3, &wEntry );

        break;
    }

}

/*
 * MatchColor( plTable, fValue ) : iEntry
 *
 *      plTable         Tabelle für Farbwerte
 *      fValue          Zu suchender Wert
 *
 * Hilfsfunktion, die eine Spectrum-Tabelle nach
 * einem bestimmten Eintrag mittels vorhandenen Schlüssels, durchsucht.
 * Ein Index zum genauesten Ebenbild wird zur weiteren Verwendung
 * zurückgeliefert, wenn zwischen Farbmischmodellen umgeschaltet wird.
 *
 * Es kann vorkommen, daß kein exaktes Ebenbild vorhanden ist (aufgrund
 * der diskreten Natur der Spektrum-Tabelle). Als Ergebnis
 * können einige Unterschiede in der resultierenden Farbe sichtbar sein.
 *
 */

static int MatchColor( plTable, fValue )
    LONG *      plTable;
    float       fValue;
{
    int         iEntry;
    int         iMinimum;
    float           fMinimum;

    /* Initialisierung */
    iMinimum = 0;
    fMinimum = ABS(*((float*)&plTable[16])-fValue);

    /* Sucht nach bestem Äquivalent */
    for ( iEntry=16; iEntry<32; iEntry++ ) {
        if ( ABS(*((float*)&plTable[iEntry])-fValue) < fMinimum ) {
            iMinimum = iEntry - 16;
            fMinimum = ABS(*((float*)&plTable[iEntry])-fValue);
        }
    }

    /* Liefert besten Eintrag zurück */
    return( iMinimum );

}
.
.
.
```

Listing 2: *(Ende)*

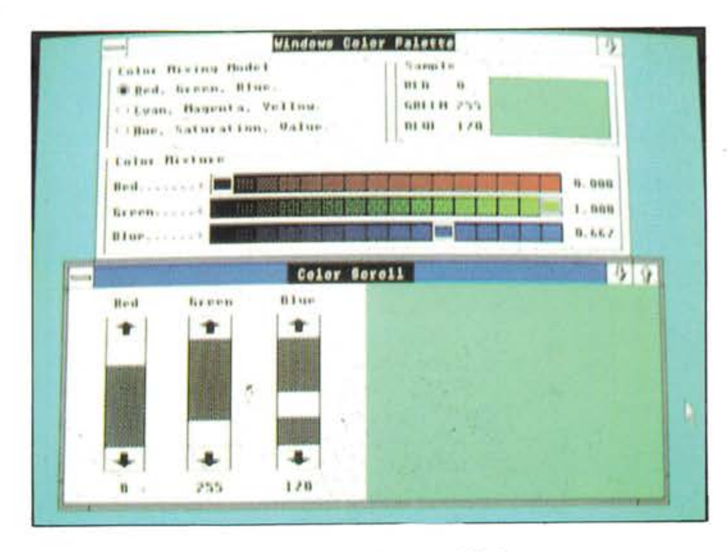


Bild 2: *Ein Vergleich von ColorScr und Palette.*

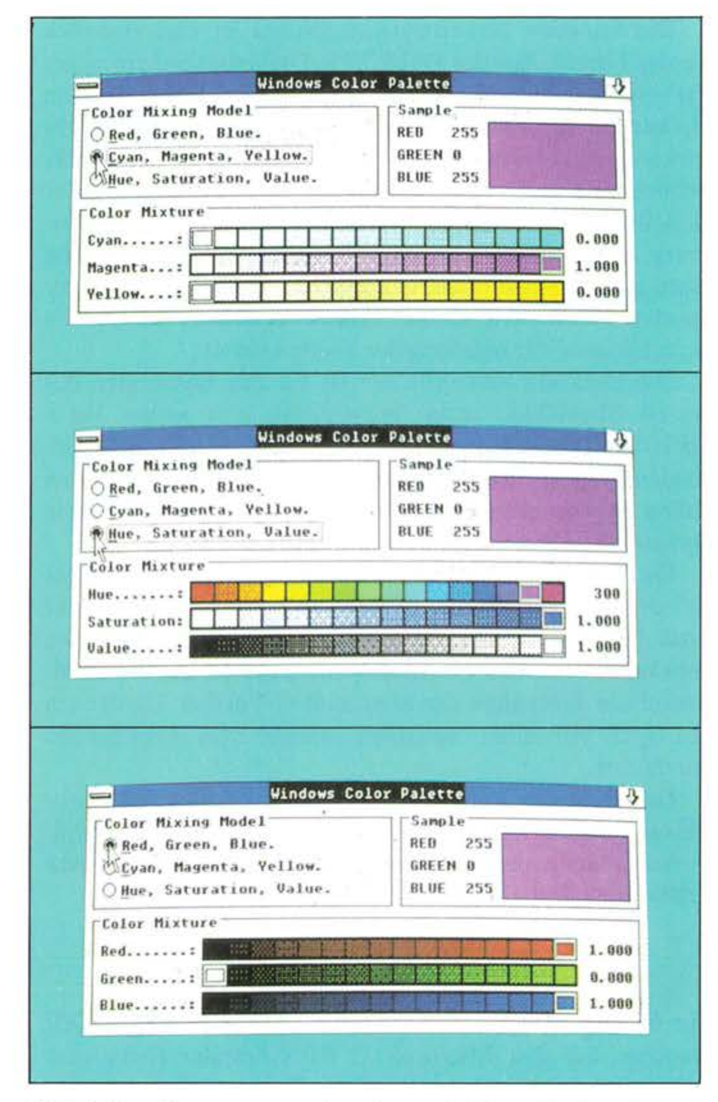


Bild 3: *Das Programm Palette kann nicht nur Farben in drei verschiedenen Modellen anzeigen, sondern es kann auch Farben und Farbmischwerte beim Umschalten zwischen den Modellen anpassen, wie mit diesen Bildschirmen gezeigt wird.*

Windows Palette

Nachdem Sie nun verstehen, wie eine anwenderdefinierte Steuerung arbeitet, wollen wir ihren Nutzen anhand einer kleinen Anwendung anschauen. Windows-Palette ist eine logische Erweiterung des Programms ColorScr von Charles Petzold (siehe MSJ März/April 1988, Seite 34). Palette ermöglicht wie ColorScr auch das Mischen von RGB-Farben zum Erstellen einer selbstdefinierten Farbmischung. Palette verwendet jedoch im Gegensatz zu ColorScr die Spectrum-Steuerung und erlaubt das Mischen mit zwei zusätzlichen Farbmischmodellen (*Bild 2*). Darüber hinaus wird beim Wechseln von einem Modell zum nächsten der Versuch unternommen, im neuen Modell die gerade definierte Farbe darzustellen (*Bild 3*).

Wenn sie PALETTE.C untersuchen, erkennen Sie die Datenstrukturen, die beim Arbeiten mit den vier Modellen verwendet werden:

CRT	GDI Rot-Grün-Blau
RGB	Rot-Grün-Blau
CMY	Cyan-Magenta-Gelb
HSV	Farbabtönung-Sättigungs-Wert (hue saturation value)

Das CRT-Modell ist direkt analog zum RGB-Modell, das vom Graphics Device Interface (GDI) unterstützt wird. Beachten Sie, daß ein Unterschied zwischen dem GDI-RGB-Modell und dem von Palette unterstützten RGB-Modell besteht. Während der Mischung einer Farbe werden die RGB-, CMY- und HSV-Werte verwendet, um einen äquivalenten RGB-GDI-Wert (in der Folge CRT genannt) herzustellen, der in der rechten oberen Ecke des Palette-Fensters dargestellt wird. Sie erfahren mehr über Farbmischmodelle in dem vorausgegangenen Artikel über Farbmisch-Grundlagen.

Das verwendete Fenster, mit dem in der Applikation gearbeitet wird, ist in PALETTE.RC als PALETTE-Dialogbox definiert. Obwohl dieses Fenster ursprünglich mit dem Dialogbox-Editor erzeugt wurde (Teil des Windows-SDK), wurde es in wichtigen Teilen manuell verändert.

Die erste Veränderung war das Hinzufügen von `WS_MINIMIZEBOX` zur Liste der Dialogbox-Stil-Attribute. In Kombination mit einer geringfügigen Änderung der Fensterklassen-Datenstruktur kann die Dialogbox zu einem Sinnbild werden.

Die zweite Änderung betraf die Schaltfläche `ID_SAMPLE`. Der Schaltflächen-Stil wurde verändert, sodaß die Schaltfläche am Anfang ausgeschaltet wird und das übergeordnete Fenster den Schaltflächeninhalt zeichnet. Dies wurde durch Veränderung des Schaltflächen-Stils zu `BS_USERBUTTON | WS_DISABLED|WS_CHILD` erreicht.

Die dritte Veränderung war das Hinzufügen dreier Instanzen der Spectrum-Steuerung zur Dialogbox-Definition. Bei der Arbeit mit dem Dialogbox-Editor werden die Größe und Position der Spectrum-Steuerung durch einen statischen Rahmen vorherbestimmt. Alle Rahmen werden später entfernt und dafür die Spectrum-Steuerung in Originalgröße eingesetzt.

Beim Start von PALETTE.C werden neun kleine Arrays definiert, die die voreingestellten Farbtabellen für das Spectrum-Steuerungsfenster enthalten. Die ersten drei Tabellen bestimmen die vorgegebenen Farbwerte für die Rot-, Grün- und Blau-Spektren. Jede Tabelle besteht aus 16 CRT-Werten, gefolgt von 16 Long-Darstellungen Ihrer Fließkommawerte. (Im theoretischen RGB-Modell liegt jeder Farbwert im kontinuierlichen Bereich {0,1}, wogegen das CRT-Modell diskrete Werte im Bereich {0,255} benutzt.) Das zweite und dritte Tabellen-Triple definiert die Standard-Werte für das CMY- bzw. HSV-Modell.

Obgleich die Spectrum-Steuerung keine obere Grenze für die Größe der Farbtabellen festlegt, muß doch jedes dargestellte Farbmuster genügend groß sein, um leicht sichtbar zu sein. In Verbindung mit dem Vorschlag für die maximale Größe von Dialogboxen aus dem »Windows Style Guide« ergibt sich eine praxisbezogene obere Grenze von etwa 30 Farbwerten.

Die Funktion `WinMain` in PALETTE.C (*Listing 2* zeigt einige Funktionen) unterscheidet sich von denen, die Ihnen bisher untergekommen sind. Im Unterschied zu einem konventionellen `WinMain` besitzt dieses keine Meldungsschleife - der Empfang und die Abfertigung von Meldungen wird dem Dialogbox-Manager überlassen. Abgesehen von der Erzeugung einer Dialogbox, ist der einzige andere Zweck dieser Routine die Registrierung der Spectrum-Steuerung durch Verwendung der aktuellen Instanzenhandle als Parameter. Falls eine vorhergegangene Instanz der Applikation existiert, wird die Spectrum-Steuerung nicht neu registriert.

Die Funktion `PaletteDlgFn` ist, wie die meisten anderen Dialog-Funktionen, für die Bearbeitung aller Meldungen verantwortlich, die mit der PALETTE-Dialogbox in Zusammenhang stehen. Die meisten dieser Funktionen bestehen aus Standard-Windows-Code, mit einigen Ausnahmen in kritischen Gebieten.

Die erste Ausnahme ist die Art, in der die Meldung `WM_INITDIALOG` verwaltet wird. In diesem Abschnitt wird das Dialogbox-Sinnbild (vorbelegt mit NULL) mit einem Aufruf der Funktion `SetClassWord` definiert. Dieser kleine Unterschied ermöglicht der Dialogbox, bei gleichzeitigem Hinzufügen von `WS_MINIMIZEBOX` zum Fenster-Stil, zu einem Sinnbild zu werden.

Die zweite Ausnahme ist die Verwaltung der Spectrum-Unterfenster-IDs bei der Bearbeitung der Meldung `WM_COMMAND`. Wie bereits bei der Besprechung der Funktion `SpectrumWndFn` erwähnt, wird ein `WM_COMMAND` (mit `wParam` gleich der Unterfenster-ID und `lParam` gleich dem zugehörigen numerischen Wert) immer dann erzeugt, wenn der Anwender eine neue Farbe aus dem Spektrum auswählt. Sobald diese Meldung von der Dialog-Funktion empfangen wird, wird der passende Farbwert (mit einer Long/Float-Umwandlung) definiert, der zutreffende Farbwert berechnet und der numerische Wert im Textfeld rechts neben der Spectrum-Steuerung abgebildet.

Das dritte kritische Gebiet ist die Verarbeitung von `ID_SAMPLEID` bei der Bearbeitung von `WM_COMMAND`. Die PALETTE-Dialogbox enthält eine anwenderdefinierte Schaltfläche. Wann immer die Schaltfläche neu gezeichnet werden muß, ist `HIWORD(lParam)` gleich `BN_Paint` und `LOWORD(lParam)` gleich der Schaltflächen-Fenster-Handle. Durch Übernahme des Bildschirm-Kontextes für diese Schaltfläche können wir eine Reproduktion der gegenwärtig gewählten Farbmischung herstellen.

Die Funktion `SelectMixingModel` ist verantwortlich für das Umschalten der PALETTE-Dialogbox auf ein neues Farbmischmodell, sobald eines gewählt wird. Dabei wird ein Update der zugehörigen Textfelder und abgebildeten Farbspektren vorgenommen, um das neue Modell zu berücksichtigen. Es sucht auch nach dem ähnlichsten Ebenbild der aktuellen Farbe im neuen Modell. Obgleich diese Angleichung normalerweise zu einer beinahe identischen Farbe führt, gibt es Situationen, in denen die Ganzzahl-Natur der Spectrum-Steuerung diesen Prozeß behindert und es zu einer leichten Veränderung der Farbe kommt.

Die Funktion `MatchColor` ist für das Bestimmen des besten Ebenbildes unter Verwendung der neuen Farb-Spektrum-Tabellen verantwortlich. Dieser Prozeß der Angleichung ist relativ zuverlässig und führt in den meisten Fällen zu vernünftigen Ergebnissen, obwohl er nur lokale Optima und Extremwertpunkte verwendet.

Die Funktion `DlgPrintf` arbeitet wie `sprintf`, außer daß die Ausgabe an eine Dialogbox-Textsteuerung geleitet wird. Für jene, denen die Strukturen in diesen Routinen unbekannt sind: Die Funktion `vsprintf` ist ein Standard-Aufruf der Bibliothek der Microsoft C Version 5.0, der ein `sprintf` mit einer variablen Anzahl von Argumenten durchführt.

Der Rest des Palette-Moduls besteht aus sechs Farbumwandlungen, die einen Farbwert in einen anderen konvertieren. Jede dieser Routinen ist im vorhergegangenen Artikel detailiert erklärt.

Programm-Erstellung

Zur Erzeugung einer ausführbaren Datei PALETTE.EXE brauchen Sie den Microsoft C 5.0 Compiler (oder eine spätere Version) und das Windows Software Development Kit ab Version 2.03. Bevor Sie die Applikation zum Laufen bringen können, müssen Sie möglicherweise einige Änderungen der LNK- und DEF-Dateien durchführen, um Ihre Festplattenstruktur zu berücksichtigen.

Nachdem Sie diese Veränderungen vorgenommen haben, können Sie PALETTE.EXE mit dem folgenden Befehl erzeugen:

```
MAKE PALETTE
```

Nach dem Kompilieren und Linken des Programms können Sie mit PALETTE auf Ihrem Computer experimentieren. Wie Sie sehen werden, ist das Entwerfen eigener Dialogbox-Steuerung eine einfache Aufgabe, die jede Anwenderschnittstelle stark erweitern kann. Mit ein wenig Mühe können Sie Spectrum auch für einige andere interessante Steuerungen verwerten. Es wird nicht lange dauern und Sie entwickeln Ihre eigenen Software-Toolboxen für das nächste interessante Projekt.

Kevin P. Welch

Fenster und Fensterklassen unter Windows (Teil 1):

Einführung in Fenster-Unterklassen

Es ist einsichtig, daß bei Microsoft Windows der Begriff des Fenster (»Window«) eine wichtige Rolle spielt. Andererseits sind die zahlreichen Möglichkeiten, die die Fensterverwaltung von Windows dem Programmierer bietet, ziemlich unbekannt, da das Windows-SDK *[1]* darauf nur am Rande und ohne weitergehende Beispiele eingeht. In diesem Artikel und einem weiteren im nächsten Heft wird detailliert auf Windows-Fenster eingegangen.

Die Anwendungs-Beispiele im *Programmer's Learning-Guide* besitzen alle den gleichen Aufbau: Einerseits ein ziemliches kleines Hauptprogramm, welches die Anwendung initialisiert und die bekannte Windows-Meldungsschleife besitzt und andererseits eine Fensterfunktion, welche alle an die Anwendung gesandten Nachrichten verarbeitet und den Inhalt des Anwendungsfensters zeichnet. Zu diesen beiden Hauptteilen kommen dann noch Unterprogramme und einige Funktionen zur Verwaltung der Dialogfelder (Dialog Boxes). Hat man als Anfänger die Struktur einer solchen Anwendung verstanden, ist man versucht, seine konkrete Programmieraufgabe in dieses Schema zu pressen. So werden dann weitere Dialogboxen hinzugefügt und die Komplexität der Anwendungsfensterfunktion steigt bei komplizierten Programmen mit umfangreicheren Zeichenaufgaben (zum Beispiel Wysiwyg-Textverarbeitung oder CAD-Programm) gewaltig an. Das Programm wird zunehmend unstrukturierter und auch kompliziert zu warten. Oft ließe sich dies vermeiden, würde man die Zeichenfläche eines Fensters in mehrere kleinere Teile aufteilen, die weitgehend unabhängig voneinander gezeichnet werden können. Diese Strukturierung wird von Windows mit Tochterfenstern (Child windows) und selbstdefinierbaren Fensterklassen hervorragend unterstützt. Womit wir beim Thema wären.

Die Aufgabe eines Fenster

Zunächst wollen wir jedoch ein wenig Abstand nehmen von der eingefahrenen Anwendungsstruktur des Windows-SDK und untersuchen, warum eigentlich Windows-Programme so merkwürdig aufgebaut sind. *Bild 2* zeigt den allgemeinen Aufbau einer Windows-Anwendung. Der Kern ist eigentlich weitgehend identisch mit einem normalen DOS-Programm. Hier erfolgen die Initialisierung, Datei-Ein-/Ausgabe, Geräteausgabe, interne Berechnungen usw. Zwei wichtige Dinge fehlen jedoch: die Verwaltung jeglicher Eingaben von Tastatur und Maus und die Ausgabe an den Bildschirm. Warum ist das so? Nun, bei Windows erfolgt die Kommunikation einer Anwendung mit dem Benutzer nicht direkt über Tastatur, Maus und Bildschirm sondern generell über Fenster. Wenn ein Fenster am Bildschirm sichtbar ist, kann der Benutzer die Ausgabe in diesem Fenster registrieren.

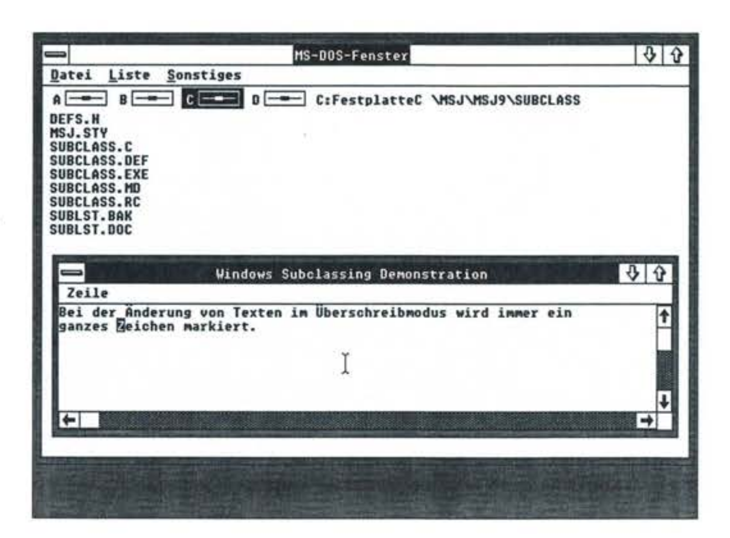
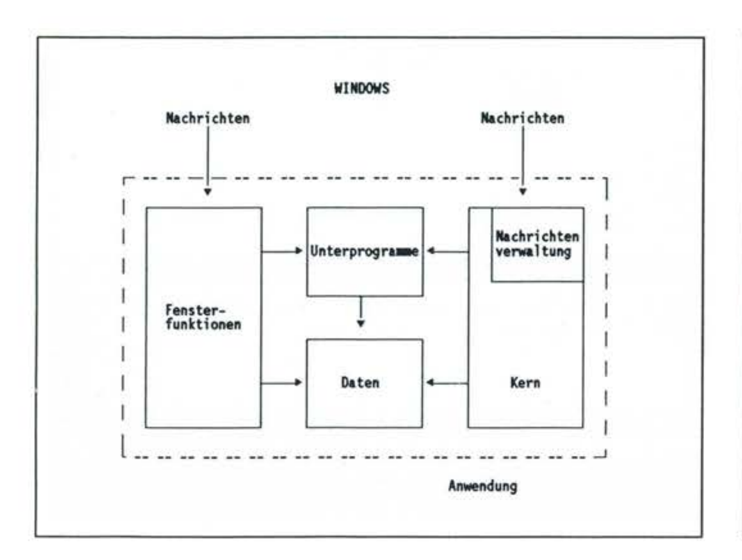


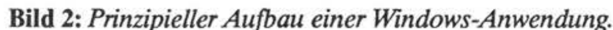
Bild 1: *Das Beispielprogramm SUBCLASS für die Demonstration von Fensterklassenerweiterungen.*

Wenn zusätzlich das Fenster den »Eingabefokus« erhalten hat, kann der Benutzer mit der Anwendung dieses Fensters über Maus und Tastatur im Dialog kommunizieren. Durch die Verlagerung der Konsolen-Ein-/Ausgabe vom Kern der Anwendung zu deren Fenstern wurde die standardisierte Ein-/Ausgabe von Windows ermöglicht, die beispielsweise einen einfachen Wechsel zwischen geladenen Anwendungen ermöglicht. Wird eine Taste gedrückt oder die Maus bewegt, schickt Windows eine entsprechende Nachricht an das Fokus-Fensters wo sie von der Fensterfunktion verarbeitet wird.

Eine weitere wichtige Aufgabe der Fensterfunktion ist das Zeichnen des Fensterinhalts (Client Area). Warum übernimmt dies nicht der Kern der Anwendung? Nun, beim Verschieben oder Vergrößern eines Fensters muß der Fensterinhalt teilweise neu gezeichnet werden. Hierzu wird dem entsprechenden Fenster die Nachricht `WM_PAINT` zugesandt und die Fensterfunktion zeichnet den Fensterinhalt teilweise neu, ohne jedoch die Bedeutung der Ausgabe zu verändern. Da dieses Zeichnen unabhängig vom Entstehungszeitpunkt der Ausgabe-Daten durchführbar sein muß, ist es sinnvoll, den Zeichenprozeß vom Kern der Anwendung in die Fensterfunktion zu verlagern und im Kern nur die Daten für die Ausgabe zu erzeugen oder zu ändern und dem Zeichenprozeß zur Verfügung zu stellen. Bei jeder Änderung der Daten fordert der Kern explizit die Fensterfunktion auf, den Fensterinhalt den Datenänderungen anzupassen. Dies geschieht im allgemeinen durch Aufruf der Funktionen `InvalidateRect` und `UpdateWindow`, beide senden im wesentlichen `WM_ERASEBKGND`- und `WM_PAINT`-Nachrichten an die Fensterfunktion.

Es gibt also eine Trennung zwischen Erzeugung und Ausgabe der Daten, im Gegensatz zu vielen DOS-Programmen, bei denen Daten zum Beispiel Zeile für Zeile aufbereitet und anschließend sofort ausgegeben werden. Jetzt werden einige Leser einwenden, daß sie noch nie den Quell-




Bild 2: *Prinzipieller Aufbau einer Windows-Anwendung.*

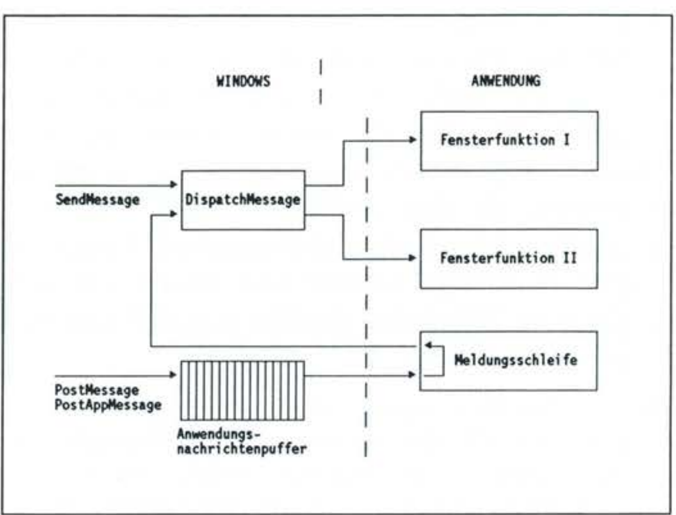


Bild 3: *Verteilung von Nachrichten innerhalb der Anwendung.*

code eines Windows-Programms gesehen haben, in dem die Ausgabedaten außerhalb der Fensterfunktion erzeugt wurden. Vielmehr geschehe meist auch die Erzeugung in der Fensterfunktion und nicht im Kern. Diese Aussage ist nicht ganz falsch, da es sich bei den meisten veröffentlichten Windows-Programmen um ausgesprochen dialogorientierte Anwendungen handelt. Hierbei werden Fensterausgaben als direkte Folge von Benutzereingaben unmittelbar erzeugt oder geändert. Da die Fensterfunktion die Eingabe verwaltet, ist es naheliegend, auch die Datenerzeugung der Fensterfunktion zu übertragen anstatt einen Umweg über den Anwendungskern zu gehen. Es sind jedoch auch Windows-Programme denkbar, die mehr im Hintergrund ablaufen und zumindest zeitweise wenig dialogorientiert sind. Hierzu zählen zum Beispiel Compiler oder Tabellenkalkulationsprogramme während der Berechnungen der Daten. Bei solchen Programmen erfolgt die Hauptarbeit bei der Datenaufbereitung im Kern und nicht bei der Ausgabe, und Ausgabeänderungen erfolgen nicht auf Anweisung des Benutzers sondern durch die Anwendung selbst. Dieser Umstand ist wichtig zu wissen, wenn man beabsichtigt, eine DOS-Anwendung auf Windows umzuschreiben: Je weniger Ein-/Ausgabe ein Programm benötigt, umso weniger muß man letztendlich neu schreiben. Andererseits: Hat man es erst einmal geschafft, einen ein-/ausgabeunabhängigen Kern einer Anwendung herauszukristallisieren, kann man ein solches Programm auch mit geringerem Aufwand an andere Grafiksysteme anpassen wie Presentation-Manager, Apple-Macintosh, UNIX-Presentation-Manager ...

Nachrichtenverwaltung im Anwendungskern

Wie man in *Bild 2* auch erkennen kann, besitzt der Kern einer Anwendung ein weiteres elementares Windows-Element, die Verwaltung der Fensternachrichten. Bis jetzt haben wir festgestellt, daß die Fensterfunktion auf den Empfang von Nachrichten ihre Aktivitäten durchführt. Doch wie gelangen die Nachrichten zu der Fensterfunktion? Hier gibt es zwei unterschiedliche Wege: Erstens kann die Nachricht direkt an das Fenster gesandt werden (und damit an die dazugehörende Fensterfunktion) wo sie sofort verarbeitet wird. Eine zweite Möglichkeit besteht darin, die Nachricht an die Anwendung des Fensters zu senden und es dem Kern der Anwendung zu überlassen, die Nachricht an das entsprechende Fenster weiterzuleiten. Hierbei wird die angesprochene Anwendung nicht sofort die Nachricht bearbeiten, sondern erst, wenn sie die Kontrolle erhält und es der Anwendungskern für erforderlich ansieht. Die Nachrichten werden deshalb zunächst in einem Puffer (Application Message Queue) abgelegt.

Im ersten Fall wird die Nachricht mit der Funktion `SendMessage` gesendet, im zweiten Fall mit `PostMessage`. Beide Übertragungsformen haben jeweils ihre Vor- und Nachteile und werden daher von Fall zu Fall verwendet. Auch Windows selbst benutzt beide Verfahren. Bei der Entwicklung einer Windows-Anwendung sollte man stets beide Möglichkeiten im Auge haben. Bei der Auswahl entscheidet sich beispielsweise, ob eine Anwendung bei der Eingabe »ruckt« oder sich »fließend« nach dem Benutzer richtet.

Die Verwaltung der Fensternachrichten ist bei den meisten veröffentlichten Windows-Anwendungen die einzige Aufgabe, die (abgesehen von der Initialisierung) der Anwendungskern vornimmt. Es handelt sich hierbei um die bekannte Windows-Meldungsschleife. Eine Ausnahme macht das Programm MAZE.EXE aus einer früheren Ausgabe des *Microsoft System Journals [2]*. Hier ist der Kern überwiegend damit beschäftigt, die Position des Balls neu zu berechnen.

Die Meldungsschleife in der vorhandenen Form übernimmt zwei Aufgaben: Einerseits übergibt sie für eine gewisse Zeit die Kontrolle an das System und andere geladene Anwendungen durch Aufruf der Funktionen `Get-`

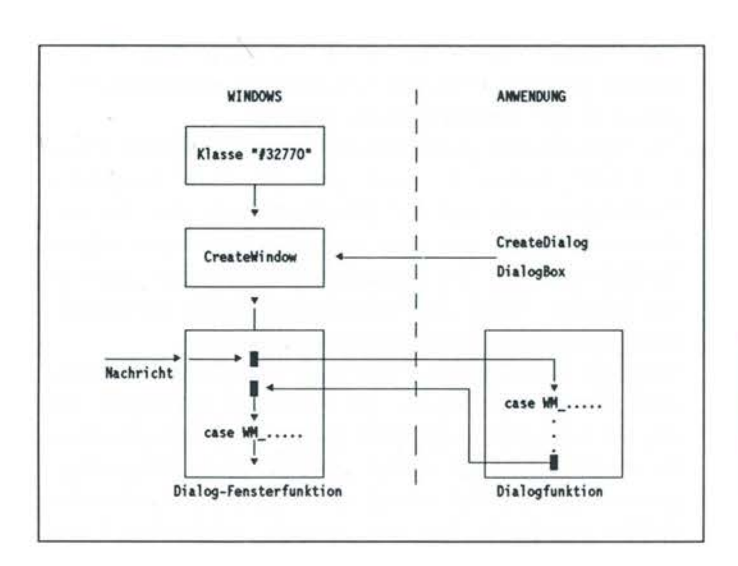
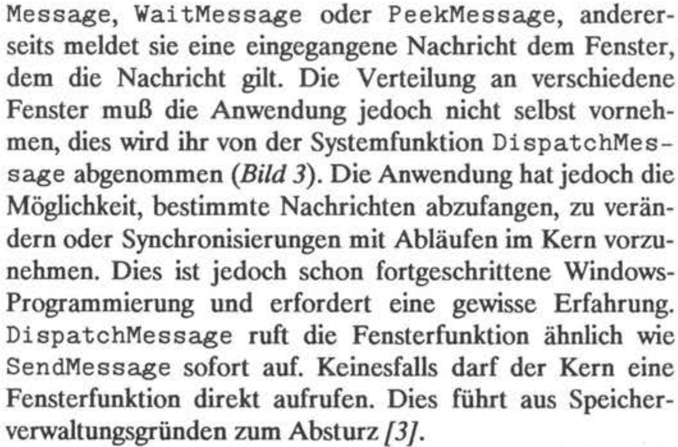



Bild 4: *Aufbau der Dialogfeld-Fensterverwaltung*



`Message`, `WaitMessage` oder `PeekMessage`, andererseits meldet sie eine eingegangene Nachricht dem Fenster, dem die Nachricht gilt. Die Verteilung an verschiedene Fenster muß die Anwendung jedoch nicht selbst vornehmen, dies wird ihr von der Systemfunktion `DispatchMessage` abgenommen (*Bild 3*). Die Anwendung hat jedoch die Möglichkeit, bestimmte Nachrichten abzufangen, zu verändern oder Synchronisierungen mit Abläufen im Kern vorzunehmen. Dies ist jedoch schon fortgeschrittene Windows-Programmierung und erfordert eine gewisse Erfahrung. `DispatchMessage` ruft die Fensterfunktion ähnlich wie `SendMessage` sofort auf. Keinesfalls darf der Kern eine Fensterfunktion direkt aufrufen. Dies führt aus Speicherverwaltungsgründen zum Absturz *[3]*.

Eine Anwendung kann mehr als ein Fenster besitzen, ja eigentlich sogar beliebig viele (bis der Speicher voll ist). Jedes angelegte Fenster weiß, zu welcher Anwendung es gehört und die Fensterverwaltung benutzt solche Information, wenn sie die Funktion `PostMessage` ausführt (ein Fenster kann nur genau einer Anwendung gehören).

Nachrichten werden fast immer an Fenster geschickt, nur selten (mit `PostAppMessage`) an die Anwendung selbst. Das gilt auch für Nachrichten, die eigentlich gar nicht für ein bestimmtes Fenster gedacht sind, sondern mehr die Anwendung selbst betreffen, so die DDE-Kommunikation oder die Anfrage vom System, ob die Windows-Sitzung beendet werden darf. Dies unterstreicht noch einmal die wichtige Bedeutung des Fensters und der Fensterfunktion in Windows.

Tochterfenster bringen Übersicht

Wie in der Einleitung erwähnt, besitzen fast alle Beispiele des SDK und auch alle bisher im *Microsoft System Journal* aufgeführten Programme nur ein einziges explizit vereinbartes Fenster, das Anwendungsfenster. Bei einfacheren

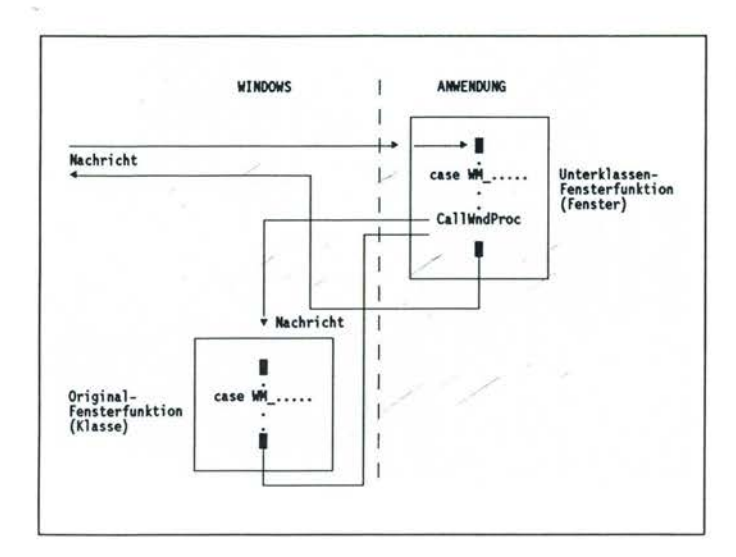


Bild 5: *Modell der Unterklassentechnik*

Anwendungen ist dies auch vollkommen ausreichend, doch bei komplizierten Programmen ist die Verwaltung beim (Neu-)Zeichnen des Fensterinhalts doch ziemlich groß und die Fensterfunktion wird stark aufgebläht. Hier bietet sich an, weiterhin ein einziges großes Anwendungsfenster zu haben, dessen Zeichenfläche aber in mehrere kleinere Fenster aufzuteilen und diese unabhängig voneinander in ihren einzelnen Fensterfunktionen zu zeichnen. Diese neuen Fenster enthalten natürlich keine Rahmen, da der Inhalt des Anwendungsfensters weiterhin als homogene Einheit dem Benutzer präsentiert werden soll. Die Fenster müssen fest mit dem Anwendungsfenster verbunden sein, wird letzteres bewegt, sollen sich die Fenster schließlich mitbewegen. Muß ein Teil des Anwendungsfensters neu gezeichnet werden, müssen auch die in diesem Teil liegenden Fenster eine Aufforderung zum Neuzeichnen erhalten.

Die Fensterverwaltung von Windows stellt dem Programmierer ein Konzept zur Verfügung, welches tatsächlich obigen Anforderungen (und einigen weiteren) genügt. Die Windows-Entwickler wählten hierzu den Begriff des Tochterfensters (im Original Child Window). Ein Tochterfenster steht im Zeichenbereich eines anderen (Vater-)Fensters. Das Vaterfenster kann beliebig viele Tochterfenster besitzen, diese können sich überlappen oder nebeneinander stehen. Bei der Mehrfach-Dokumentenschnittstelle (Multi-Document-Interface) *[1]* werden sogar Tochterfenster verwendet, die Rahmen zum Verschieben und Vergrößern besitzen wie normale Anwendungsfenster. Folgendes sind die wesentlichen Eigenschaften von Tochterfenstern:

- Sie können ein beliebiges Aussehen haben, also auch Rahmen oder Titelleiste besitzen, dürfen aber keine Menüleiste haben.
- Sie müssen innerhalb des Zeichenbereichs des Vaterfensters stehen, überstehende Teile werden abgeschnitten (clipping). Wird das Vaterfenster zum Sinnbild verkleinert, werden alle seine Tochterfenster unsichtbar.
- Ein Tochterfenster existiert nur solange wie das Vaterfenster existiert. Wird das Vaterfenster vernichtet, werden auch alle Tochterfenster vernichtet.
- Das Vaterfenster überwacht das Neuzeichnen der Tochterfenster, indem es diese mit den dazu benötigten Nachrichten versorgt. Bei Überlappungen von Tochterfenstern existiert eine vom Programmierer beinflußbare Zeichenlogik wie die überlappten Fensterteile gezeichnet werden. Wird das Vaterfenster selbst gezeichnet, werden dabei die Tochterfenster ausgespart.
- Wird die Maus bewegt, wird vom Vaterfenster automatisch dem Tochterfenster die Nachricht zugesandt, auf das im Augenblick die Maus zeigt. Mit Tochterfenstern lassen sich somit auf höchst einfache Weise Anwendungen realisieren, bei denen das Anklicken bestimmter Felder spezifische Aktionen bewirkt. Wird ohne Tochterfenster gearbeitet, muß jedesmal in der Anwendung explizit aus der Mausposition algorithmisch das entsprechende Feld berechnet werden.
- Tochterfenster sollten im Normalfall eine Identifizierungs-Nummer (Id) besitzen, mit der sie angesprochen werden. Da der Wert vom Programmierer beliebig gewählt werden kann (im Gegensatz zum Fensterbezug (Handle), dessen Wert von Windows vorgegebenen wird), kann er auch als Index in Felder usw. verwendet werden.

Tochterfenster können auf verschiedenste Art benutzt werden. Auch das MS-DOS-Fenster wurde mit Tochterfenstern realisiert. Wie der Leser weiß, wird in diesem Fenster eine Liste der verfügbaren Laufwerke, der aktuelle Pfad und die Tabelle der Dateien im aktuellen Verzeichnis angezeigt. Anstatt nun ein einziges großes schwerfälliges Fenster für die Darstellung zu wählen, hat man die drei verschiedenen Darstellungen in drei verschiedene Tochterfenster verteilt, deren Vaterfenster das MS-DOS-Fenster ist. Hierbei kann der Benutzer auf Anhieb nicht sehen, daß es tatsächlich drei verschiedene Fenster sind, denn die verwendeten Tochterfenster besitzen keine Rahmen und können auch nicht vom Benutzer verschoben werden.

Jedem Fenster seine Fensterklasse

Nachdem wir nun wissen, welche Bedeutung Fenster für eine Anwendung besitzen, wird es Zeit, sich mit der Entstehung eines Fensters zu befassen. »Kein Problem«, werden manche sagen, »das geht mit `CreateWindow`«. Doch das ist nur die halbe Wahrheit. Mit `CreateWindow` wird ein Fenster angelegt, dazu muß man jedoch dessen Fensterklasse kennen. Diese wird vorher mit der Funktion `RegisterWindowClass` definiert, mit irgendwelchen merkwürdigen Parametern ...

Ein weiteres Mal wurde blind etwas aus Beispielen übernommen, ohne daß man sich eigentlich genauer damit beschäftigt hätte. Dabei ist das Konzept der Fensterklasse eigentlich eine der interessanten und mächtigsten Techni-

WNDCLASS.lpszClassName
: ist der Name der Klasse, angegeben als mit 0 beendete Zeichenkette.

WNDCLASS.style, Index GCW_STYLE
: ist ein Bitfeld, in dem folgende Flags gesetzt werden können:

CS_BYTEALIGNWINDOW
Das Fenster beginnt in X-Richtung so, daß es an einer Byte-Grenze im Bildschirmspeicher beginnt. Durch eine entsprechende relative Ausrichtung der Textausgabe- oder Punktbild-Positionen innerhalb des Fensters (Vielfaches von 8) läßt sich der Fensteraufbau erheblich beschleunigen.

CS_BYTEALIGNCLIENT
Wie CS_BYTEALIGNWINDOW, die Byte-Grenzen-Ausrichtung bezieht sich jedoch auf den Fensterinhalt. Sinnvoll bei Fenster, deren Rahmenbreite nicht als Konstante vorgegeben ist.

CS_SAVEBITS
Das Punktbild (Bitmap) des kompletten Fensterinhalts wird gesichert. Dadurch läßt sich das Neuzeichnen bei der WM_PAINT-Nachricht erheblich beschleunigen. Bei größeren Fenstern wird jedoch sehr viel Speicher belegt.

CS_NOCLOSE
Der SCHLIESSEN-Befehl im Steuermenü ist gesperrt.

CS_NOKEYCVT
Nicht dokumentiert, Bedeutung unklar.

CS_PARENTDC
Jedes Fenster erhält den Zeichenkontext (Display Context) des Vaterfensters.

CS_CLASSDC
Die Klasse enthält einen eigenen Zeichenkontext, d.h. alle Fenster dieser Klasse teilen sich einen gemeinsamen Zeichenkontext.

CS_OWNDC
Jedes Fenster erhält seinen eigenen Zeichenkontext (Display Context). Diese Option ermöglicht ein einfaches Verwalten der Zeichenwerkzeuge (Zeichensatz, Strichbreite), benötigt aber ca. 800 Bytes zusätzlich pro Fenster.

CS_OEMCHARS
Die Funktion TranslateMessage wandelt virtuelle Tasten nicht in ANSI-Zeichen, sondern in OEM-Zeichen um.

CS_DBLCLKS
Bei Doppelklick Meldung an das Fenster, sonst nicht.

CS_NOKEYCVT
Nicht dokumentiert, Bedeutung unklar.

CS_HREDRAW
Wird das Fenster in der Höhe geändert, wird der gesamte Bildschirminhalt für ungültig erklärt. Wichtig für Fenster, die den Inhalt größenabhängig zeichnen (zum Beispiel die Anwendung UHR).

CS_VREDRAW
Wird das Fenster in der Breite geändert, wird der gesamte Bildschirminhalt für ungültig erklärt. Anwendung wie bei CS_HREDRAW.

WNDCLASS.hInstance, Index GCW_HMODULE
: gibt den Instanzbezug der Fensterklasse an, das heißt zu wessen Modul die Fensterklasse gehört.

WNDCLASS.lpfnWndProc, Index GCL_WNDPROC
: ist die Initialisierungs-Adresse der Fensterfunktion für alle Fenster der Klasse. Dieser Wert wird bei CreateWindow dem Fenster zugewiesen (Index GWL_WNDPROC).

WNDCLASS.cbClsExtra, Index GCW_CBCLSEXTRA
: gibt an, wieviel Bytes beim Anlegen der Fensterklasse zusätzlich zum Ablegen von Fensterklassen-Daten reserviert werden sollen. Im Gegensatz zu GCW_CBWNDEXTRA ist dieser Wert nicht sehr nützlich und sollte daher im allgemeinen 0 sein.

WNDCLASS.cbWndExtra, Index GCW_CBWNDEXTRA
: gibt an, wieviel Bytes beim Anlegen eines Fensters zusätzlich zum Ablegen von Fenster-Daten reserviert werden sollen. In dem angelegten Bereich können lokale Fensterdaten abgelegt werden (siehe 2. Teil dieses Artikels). Wenn pro Anwendung nur ein Fenster pro Klasse angelegt wird (wie bei dem Hauptfenster der Anwendung), ist dieser Wert normalerweise 0.

WNDCLASS.hbrBackground, Index GCW_HBRBACKGROUND: bezeichnet die Farbe oder das Farbmuster (Brush) des Fensterhintergrunds. Dieser Wert wird verwendet, wenn die Nachricht WM_ERASEBKGND das Fenster löscht und dieses Löschen nicht explizit die Fensterfunktion durchführt.

WNDCLASS.hCursor, Index GCW_HCURSOR
: ist der Bezug auf den Mauszeiger des Fensters. Immer wenn der Benutzer die Maus in den Fensterbereich verschiebt, wird dieser Zeiger angezeigt. Wird kein Bezug angegeben (NULL), muß die Fensterfunktion jedesmal den Mauszeiger selbst setzen, sobald sie eine Nachricht erhält, daß die Mausposition innerhalb ihrer Fenstergrenzen liegt.

WNDCLASS.lpszMenuName, Index GCL_MENUNAME
: ist der Name oder die Ressourcen-Nummer des Menüs, das bei der Anlegung eines Fensters diesem als Initialisierung zugewiesen wird. Wird kein Menü angegeben, erhält entsprechend das Fenster zunächst kein Menü. Tochterfenster können kein Menü besitzen.

WNDCLASS.hIcon, Index GCW_HICON
: ist der Bezug auf das verwendete Sinnbild. Braucht nur angegeben zu werden, wenn das Fenster auf Sinnbildgröße verkleinert werden kann. Wird kein Sinnbild angegeben, muß ggf. das Fenster sein eigenes Sinnbild zeichnen (wie die Anwendung UHR.EXE).

Tabelle 1: *Die Fensterklassen-Parameter*

ken, die Windows dem Programmierer zur Verfügung stellt. Deshalb wollen wir uns im weiteren detailliert mit Fensterklassen beschäftigen.

Jedes Fenster besitzt bestimmte Eigenschaften, die durch Parameter beschrieben werden. Dazu gehören neben der Größe und dem Aussehen des Fensters auch die Besitzverhältnisse und die Adresse der Fensterfunktion. Anstatt nun diese Parameter bei jedem Anlegen eines Fensters erneut anzugeben, faßt man einige Eigenschaften zu einer Fensterklasse zusammen und definiert zunächst diese. Sie besitzt im Gegensatz zu einem Fenster einen Namen und die Definition erfolgt mit `RegisterWindowClass`. Die Parameter einer Fensterklasse sind in *Tabelle 1* aufgeführt und kurz beschrieben. Andere Parameter sind dagegen dem Fenster direkt zugeordnet. Diese Parameter sind entsprechend in *Tabelle 2* beschrieben.

Warum haben die Entwickler von Windows nun zwischen Fenstern und Fensterklassen unterschieden? Dies hat mehrere Gründe:

- Oft werden mehrere Fenster angelegt, aber die Eigenschaften der Fenster untereinander unterscheiden sich nicht stark, zum Beispiel nur in Position, Größe und Fensterinhalt. Doch dann können sich alle Fenster einer Klasse eine gemeinsame Fensterfunktion teilen oder besitzen diesselbe Hintergrundfarbe oder Mauszeiger (Cursor). Es wäre Speicherverschwendung, würden gemeinsame Fensterdaten mehrfach gehalten.
- Es gibt vordefinierte Fensterklassen (Static, Button, Edit, ListBox, ScrollBar), die von Windows angelegt wurden und dazu verwendet werden, komplizierte aber häufig benötigte Dialogelemente zu implementieren. So kann beispielsweise durch Verwendung der EDIT-Klasse mit einfachen `CreateWindow`-Aufrufen sowohl die Zeileneingabe als auch ein kompletter Texteditor realisiert werden. Der Programmierer braucht das Rad nicht ständig neu zu erfinden.
- Klassen von in Anwendungen häufig benötigten Fenstern werden unabhängig von der Anwendung der Fenster entwickelt und implementiert. Die Implementierung des Fensters (Initialisierung und Fensterfunktion) kann in eine anbindbare Bibliothek (dynamic link library, DLL) verlagert werden und erweitert damit die Liste der vordefinierten Fensterklassen.
- Selbstdefinierte Fenster kann man auch in Dialogfeldern (Dialog boxes) darstellen. Dazu wird einfach der Klassenname bei der Beschreibung des Dialogfelds in der Ressourcen-Datei als Argument der CONTROL-Anweisung angegeben. Man ist nicht länger auf die Standard-Dialogelemente in Dialogfeldern angewiesen.

Fensterklassen sind hervorragend dazu geeignet, Anwendungen zu strukturieren. Die Entwicklung eines Fensters geschieht bei der Spezifikation der Klasse. Es wird ein Satz von Nachrichten für das Fenster zusammengestellt, mit denen der Fensterinhalt verändert werden kann.

Index GWL_WNDPROC
: ist die Adresse der Fensterfunktion. Bei CreateWindow wird ihr der Adressenwert der Fensterklasse zugewiesen (Index GCL_WNDPROC). Der Wert wird bei Fenster-Unterklassen geändert.

Index GWW_HINSTANCE
: gibt den Instanzenbezug des Fensters an, das heißt, zu welchem Modul das Fenster gehört. Dieses muß nicht zwangsläufig dasselbe Modul sein, das die Fensterklasse angelegt hat (Index GCW_HMODULE).

Index GWW_HWNDPARENT
: ist der Bezug auf das Vaterfenster des Fensters. Ein Wert von 0 bezeichnet, daß das Fenster auf oberster Ebene liegt und nur den Bildschirmhintergrund als Vater hat. Tochterfenster haben hier einen von 0 verschiedenen Wert.

Index GWW_HWNDTEXT
: ist der Bezug auf den Fenstertext. Doch dieser Bezug darf nur von Windows intern verwendet werden. Um den Fenstertext zu erhalten oder zu ändern, müssen die Funktionen GetWindowText und SetWindowText verwendet werden.

Index GWW_ID
: ist die Bezeichnung eines Tochterfensters. Sie wird zur Unterscheidung mehrerer Fenster einer Klasse verwendet, zum Beispiel beim Versenden von Nachrichten an ein bestimmtes Fenster. Insbesondere wird dieser Wert innerhalb von Dialogfeldern (Dialog Boxes) verwendet.

Index GWL_STYLE
: ist der Stil des Fensters. Bei diesem 32-Bit-Wert sind nur in den oberen 16 Bits die allgemeinen Fensterstile (Rahmenstil, Tochterfenster-Verhalten usw.) enthalten. In den unteren 16 Bits können dagegen klassenspezifische Stile angegeben werden. Sowohl der Fensterstil als auch der Stil der vordefinierten Fensterklassen ist in *[1]* bei der Beschreibung der Funktion CreateWindow ausführlich beschrieben. Beim Entwickeln eigener Fensterklassen sollten wichtige Darstellungsstile in den unteren 16 Bits definiert werden, da dann diese Stile auch bei einer Dialogfeld-Definition in der Ressourcen-Datei festgelegt werden können.

Tabelle 2: *Die Fenster-Parameter*

Anschließend wird die Fensterfunktion implementiert, die diese neuen Nachrichten und die Standardnachrichten (Größe, Neuzeichnen, etc.) verarbeitet. Nun ist die neue Fensterklasse fertig und kann genauso verwendet werden wie die vordefinierten Fensterklassen EDIT, BUTTON etc. Die Anwendung der neuen Klasse erfolgt genauso einfach wie bei den vordefinierten Klassen, entweder explizit mit CreateWindow oder implizit durch Verwendung in Dialogfeld-Definitionen.

```
/*****************************************************************
 ---- S U B C L A S S ---- MS-Windows Application ----------
==================================================================
This MS-Windows application is a small program for the
demonstration and study of the Windows programming technique of
window subclassing.
------------------------------------------------------------------
Copyright 1988 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 88-Dec-07 --- All rights reserved.
*****************************************************************/

/* define/undefine non-debugging option name */
#undef  NDEBUG

#define  NOMINMAX
#include <WINDOWS.H>
#include "DEFS.H"

/* further C standard headers */
#include <stdlib.h>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)

/*****************************************************************
 -------------------- Application Variables -----------------
*****************************************************************/
/* application name */
char  *rzAppName = "SUBCLASS";

/* handle to application instance */
HANDLE hiMain;

/* --------------
   window handles
   --------------*/
HWND hwMain;
HWND hwEdit;

/* -----------------------------------
   window procedure instance addresses
   -----------------------------------*/
FARPROC rfwSubEdit;
FARPROC rfwOrgEdit;
FARPROC rfdLine;


/*****************************************************************
 --------------- "EDIT" Subclassing function ----------------
*****************************************************************/

/*****************************************************************
 f w S u b E d i t
==================================================================
/// Subclassing window function ///
This function adds a overwrite option and
the "Copy" command to the "EDIT" class.

Parameters:
   standard message data (see fwMain)
Return:
   standard window function return value.
*****************************************************************/

LONG FAR PASCAL fwSubEdit(HWND hw,WORD iMsg,WORD uP1,
                          DWORD ulP2)
 {static BOOL bOverWrite=FALSE;
  BOOL bShiftDown; /* TRUE if SHIFT pressed */
  BOOL bCtrlDown;  /* TRUE if CTRL pressed */
  LONG vRetValue;
  DWORD uSelRange;
```

Listing 1: *Das Programm SUBCLASS.C*

```
  switch (iMsg)
   {case WM_SETFOCUS:
      /* gets input focus: clear overwrite mode */
      bOverWrite=FALSE; break;
    case WM_KEYDOWN:
      if (ulP2&(1<<29)) break; /* don't consume system key */
      bShiftDown=GetKeyState(VK_SHIFT)>>15;
      bCtrlDown=GetKeyState(VK_CONTROL)>>15;
      if (uP1==VK_INSERT)
       {/* "Ins": analyse context */
        if (bCtrlDown)
         {/* CTRL+INS: set selection into clipboard */
          iMsg=WM_COPY; break;
         }
        else if
         (!bCtrlDown && !bShiftDown &&
          GetWindowLong(hw,GWL_STYLE)&ES_MULTILINE
         )
         {/* toggle overwrite mode */
          bOverWrite=!bOverWrite;
          if (!bOverWrite)
           {/* exact one character selected => unselect this */
            uSelRange=
              CallWindowProc(rfwOrgEdit,hw,EM_GETSEL,0,0L);
            if (HIWORD(uSelRange)-LOWORD(uSelRange)==1)
             {/* select no character,
                  don't change caret location */
              CallWindowProc
               (rfwOrgEdit,hw,EM_SETSEL,
                0,MAKELONG(LOWORD(uSelRange),LOWORD(uSelRange))
               );
             } /* if */
           } /* if */
          vRetValue=0L; goto CheckSel; /* key is consumed */
         } /* if */
       } /* if */
   } /* switch */
  vRetValue=CallWindowProc(rfwOrgEdit,hw,iMsg,uP1,ulP2);
  CheckSel:
  if (bOverWrite &&
   (iMsg==WM_KEYDOWN||iMsg==WM_CHAR||iMsg==WM_LBUTTONUP))
   {/* overwrite mode:
       check if at least one character is selected */
    uSelRange=CallWindowProc(rfwOrgEdit,hw,EM_GETSEL,0,0L);
    if (LOWORD(uSelRange)==HIWORD(uSelRange))
     {/* select location character */
      /* normally: caret at left of selection */
      int iStart=1,iEnd=0;
      if (iMsg==WM_KEYDOWN && uP1==VK_RIGHT && bShiftDown)
       {/* expand selection to right: set caret at right */
        iStart=-1;
       } /* if */
      CallWindowProc
       (rfwOrgEdit,hw,EM_SETSEL,
        0,MAKELONG(LOWORD(uSelRange)+iStart,LOWORD(uSelRange))
       );
     } /* if */
   } /* if */
  return vRetValue;
 } /* fwSubEdit() */

/*****************************************************************
 -------------------- Dialog box functions ------------------
*****************************************************************/

/*****************************************************************
 f d L i n e
==================================================================
/// Dialog Box function ///
This function processes any messages received
by the DBX_LINE dialog box.

Parameters:
   standard message data (see fwMain)
Return:
   standard dialog box function value. DialogBox() returns TRUE.
*****************************************************************/
```

Listing 1: *(Fortsetzung)*

```
BOOL FAR PASCAL fdLine(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {int i;

  switch (iMsg)
   {case WM_INITDIALOG:
      /* subwindowing of IDLINE "edit" window */
      SetWindowLong
       (GetDlgItem(hw,IDLINE),GWL_WNDPROC,(LONG)rfwSubEdit);
      return TRUE;
    case WM_COMMAND:
      if (uP1==IDOK)
       {/* close dialog box */
        EndDialog(hw,TRUE);
        return TRUE;
       } /* if */
      break;
   } /* switch */
  return FALSE;
 } /* fdLine() */


/*************************************************************
 -------------------- Main Window function -------------------
*************************************************************/

LONG FAR PASCAL fwMain(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {switch (iMsg)
   {case WM_CREATE:
      /* set application window handle */
      hwMain=hw;
      return 0L;
    case WM_SIZE:
      /* set new size of EDIT client window */
      MoveWindow(hwEdit,0,0,P2LO,P2HI,FALSE);
      return 0L;
    case WM_COMMAND:
      if (uP1==CMD_LINE)
       {/* dialog box for testing line */
        DialogBox
         (hiMain,MAKEINTRESOURCE(DBX_LINE),hwMain,rfdLine);
        SetFocus(hwEdit); /* focus to edit field again */
        return 0L;
       } /* if */
      break;
    case WM_CLOSE:
      DestroyWindow(hw);
      return 0L;
    case WM_DESTROY:
      PostQuitMessage(0); return 0L;
   } /* switch */
  return DefWindowProc(hw,iMsg,uP1,ulP2);
 } /* fwMain() */


/*************************************************************
 -- Window dependent initialization (one for all instances) --
*************************************************************/

/*************************************************************
 C r e a t e C l a s s
=============================================================
The classes of all application specified windows are
registered.

Parameters:
 none
Return:
 TRUE   if all classes created.
 FALSE  if any error during creation (memory error).
*************************************************************/

BOOL CreateClass(VOID)

 {WNDCLASS wndclass;
```

Listing 1: *(Fortsetzung)*

```
  /* --- Create window class of Application --- */

  wndclass.lpszClassName=rzAppName;
  wndclass.hInstance=hiMain;
  wndclass.lpfnWndProc=fwMain;
  wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
  wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
  wndclass.style=CS_HREDRAW|CS_VREDRAW|CS_BYTEALIGNCLIENT;
  wndclass.hIcon=NULL;
  wndclass.lpszMenuName=MAKEINTRESOURCE(MNU_MAIN);
  wndclass.cbClsExtra=0; wndclass.cbWndExtra=0;
  /* register window, return if error */
  if (!RegisterClass(&wndclass)) return FALSE;
 } /* CreateClass() */


/*************************************************************
 -- Instance dependent initialization (for every instance) --
*************************************************************/

/*************************************************************
 S e t G l o b a l P r o c e d u r e
=============================================================
All entry procedures (called by Windows systems) get a calling
instance here. This is done by the MakeProcInstance() function.

Parameters:
 none
Return:
 TRUE   is returned if all procedures data set.
 FALSE  if returned if any error occured (memory too small).
*************************************************************/

BOOL SetGlobalProcedure(VOID)

 {rfdLine=MakeProcInstance(fdLine,hiMain);
  if (rfdLine==NULL) return FALSE;
  rfwSubEdit=MakeProcInstance((FARPROC)fwSubEdit,hiMain);
  if (rfwSubEdit==NULL) return FALSE;
  return TRUE;
 } /* SetGlobalProcedure() */


/*************************************************************
 C r e a t e A p p W i n d o w
=============================================================
All global existing windows are created by this function.

Parameters:
 none
Return:
TRUE is returned if all data read, otherwise FALSE
(memory too small).
*************************************************************/

BOOL CreateAppWindow(VOID)

 {/* application's window, return if error */
  if (!CreateWindow
      (rzAppName,"Windows Subclassing Demonstration",
       WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,0,CW_USEDEFAULT,0,
       NULL,NULL,hiMain,NULL
      )
     )
    return FALSE;
  hwEdit=CreateWindow
   ("EDIT",NULL,WS_CHILD|WS_VISIBLE|ES_MULTILINE|WS_VSCROLL|
    WS_HSCROLL|ES_AUTOHSCROLL|ES_AUTOVSCROLL,
    0,0,0,0,hwMain,IDEDIT,hiMain,NULL
   );
  if (hwEdit==NULL) return FALSE;
  /* redirect "EDIT" class function */
  rfwOrgEdit=
   (FARPROC)SetWindowLong(hwEdit,GWL_WNDPROC,(LONG)rfwSubEdit);
  return TRUE;
 } /* CreateAppWindow() */
```

Listing 1: *(Fortsetzung)*

```
/*************************************************************
 I n i t I n s t a n c e : The main init module entry point ///
=============================================================
The complete initialization of a new instance of the appli-
cation window. If no previous instance exists, the application
window is initialized and the common data for all instances
are created. Otherwise, common data are copied from the
previous instance to the new instance. All individual data in
the instance data segment are initialized. If memory is too
small, a error message box is displayed and FALSE is returned.
Parameters:
 hiNew      is the handle to the new instance of window.
 hiPrev     is the handle to the first instance of window.
 rzCmdLine  points to the command line buffer.
 vCmdShow   is the entry style of window for ShowWindow().
Return:
 TRUE is returned if initialization complete,
 otherwise FALSE (memory too small).
*************************************************************/

BOOL InitInstance(HANDLE hiNew,HANDLE hiPrev,
                  LPSTR rzCmdLine,int vCmdShow)

{hiMain=hiNew; /* set instance global */
 if(!hiPrev)
  {/* initialize first instance, exit if error */
   if (!CreateClass()) goto MemError;
  } /* if */
 /* create procedure instance addresses */
 if (!SetGlobalProcedure()) goto MemError;
 /* create all global application windows */
 if (!CreateAppWindow()) goto MemError;
 /* show created main window */
 ShowWindow(hwMain,vCmdShow); UpdateWindow(hwMain);
 SetFocus(hwEdit); /* focus to edit field */
 return TRUE;
 MemError:
 /* error: memory too small */
 return FALSE; /* return with error */
} /* InitInstance() */

/*************************************************************
 ---------------------- Main function -----------------------
*************************************************************/

WORD PASCAL WinMain(HANDLE hiNew,HANDLE hiPrev,
                    LPSTR rzCmdLine,int vCmdShow)
{MSG wm;
 /* Initialize (window and) instance, return if error */
 if (!InitInstance(hiNew,hiPrev,rzCmdLine,vCmdShow))
   return FALSE;
 /* --- application execution loop --- */
 while (GetMessage(&wm,NULL,0,0))
  {TranslateMessage(&wm); DispatchMessage(&wm);
  } /* while */
 return wm.wParam;
} /* WinMain() */
```

Listing 1: *(Ende)*

```
CC=CL >$*.ERR -c -AS -Gsw -Od -W2 -Zdp $*.C

SUBCLASS.RES: SUBCLASS.RC DEFS.H
 RC -r $*

SUBCLASS.OBJ: SUBCLASS.C DEFS.H
 SETDEBUG $* ON
 $(CC)

SUBCLASS.EXE: SUBCLASS.OBJ SUBCLASS.RES SUBCLASS.DEF
 LINK4 $*/A:16,/LI/M,,SLIBW+SLIBCEW/NOD,$*
 MAPSYM $*
 RC $*.RES
```

Listing 2: *Die Make-Datei SUBCLASS.MD*

```
/*************************************************************
 --- S U B C L A S S ---- MS-Windows Application ------------
                          Resource file
=============================================================
Copyright 1988 by
        Marcellus Buchheit, Buchheit software research
        Zaehringerstrasse 47, D-7500 Karlsruhe 1
        Phone (0) 721/37 67 76    (West Germany)
All rights reserved --- Release 1.00 of 88-Dec-07
*************************************************************/

#include <STYLE.H>
#include "DEFS.H"

/* application's menu specification
 */
MNU_MAIN MENU BEGIN
  MENUITEM "Zeile",CMD_LINE
END

/* dialog boxes specification
 */
DBX_LINE DIALOG 25,20,148,40
CAPTION "Zeilentest"
STYLE WS_CAPTION|WS_POPUP
BEGIN
 CONTROL "",IDLINE,"edit",ES_AUTOHSCROLL|ES_LEFT|WS_BORDER|
        WS_TABSTOP|WS_CHILD,4,4,140,12
 CONTROL "&Ok",IDOK,"button",WS_CHILD|WS_TABSTOP|WS_GROUP|
        BS_DEFPUSHBUTTON,49,20,50,14
END /* DBX_LINE */
```

Listing 3: *Die Ressourcen-Datei SUBCLASS.RC*

```
/* menu commands */
#define CMD_LINE        100
/* dialog boxes */
#define DBX_LINE        500
/* menu */
#define MNU_MAIN        900
/* dialog box entries */
#define IDLINE          2000
#define IDEDIT          2001
```

Listing 4: *Gemeinsame Definitionen von SUBCLASS.RC und SUBCLASS.C*

```
NAME SUBCLASS
REALMODE
EXETYPE WINDOWS
DESCRIPTION 'Window subclassing demo'
STUB '..\WINSTUB.EXE'
CODE MOVABLE DISCARDABLE SHARED
DATA MOVABLE MULTIPLE
HEAPSIZE 2048
STACKSIZE 4096

EXPORTS
 fwMain @1
 fwSubEdit @2
 fdLine @3
```

Listing 5: *Die Linker-Definitionsdatei SUBCLASS.DEF*

Trennung zwischen Entwicklung und Anwendung von Fenstern

Mit den Fensterklassen hat Microsoft die allgemeine Modul- oder Klassentechnik von Programmiersprachen wie Modula II *[8]* oder C++ *[9]* auf Fenster übertragen. Die seit langem bekannten Vorteile dieser Technik können damit auf die Entwicklung von Windows-Anwendungen Einfluß finden, auch wenn bis heute (leider) noch kein C++-Compiler Windows unterstützt.

Beim Anlauf eines größeren Software-Projekts ist es oft noch nicht ganz klar, wie die einzelnen Dialogelemente am ergonomisch günstigsten plaziert werden. Die Objekte selbst sind dagegen entweder von früheren Projekten schon vorhanden oder in ihrer Form und Funktion bereits zu einem frühen Projektstadium festlegbar. Somit kann die Entwicklung dieser Objekte schon sehr früh beginnen. Bei einer sauberen Planung steckt die Hauptarbeit in der Entwicklung und Implementierung der Dialogelemente, nicht aber in der Plazierung der Objekte oder der Verarbeitung der Nachrichten dieser Objekte, ihrer Anwendung also. Wie einfach lassen sich die vordefinierten Fensterklassen verwenden, aber welche Komplexität steckt beispielsweise hinter der Implementierung der Klasse LISTBOX. Auch kann die Entwicklung eines Projekts aufgeteilt werden in Mitarbeiter, die Fensterklassen entwickeln und mehr systemorientiert arbeiten, und Mitarbeitern, die Fenster mit diesen Klassen anlegen und dabei mehr anwenderorientiert und ergonomisch denken.

Weiter oben habe ich erwähnt, daß das MS-DOS-Fenster drei Tochterfenstern besitzt. Sie tragen die Klassennamen Disk, Dir und Path. Es ist einsichtig, daß Microsoft diese Fensterklassen auch für andere Aufgaben verwenden kann oder leicht das Aussehen des MS-DOS-Fensters durch eine andere Anordnung der Fenster verändern kann. Wie kompliziert wäre dagegen das alles, hätte man versucht, das MS-DOS-Fenster als ein riesiges Fenster ohne Tochterfenster oder ohne mehrere Fensterklassen zu implementieren!

Dialogfelder: Ein Fenster mit Tochterfenstern

Mit Dialogfeldern kann man mit wenig Aufwand Dialogelemente in Fenstern anordnen, entweder durch Angabe der Elementgröße und -position in der Ressourcen-Datei oder interaktiv durch Verwendung des Programms DIALOG.EXE. Doch was sind eigentlich Dialogfelder? Nun, das Dialogfeld selbst ist ein gewöhnliches Fenster und die Dialogeinträge sind die Tochterfenster dieses Dialogfensters. Mit dem DIALOG-Eintrag in der Ressourcen-Datei beschreibt man die Art und Größe des Dialogfensters und mit den CONTROL-Einträgen (oder CTEXT, RADIOBUTTON etc.) die Art, Größe und Position der einzelnen Tochterfenster. Die Art der Dialogelemente wird dabei einerseits durch den Klassennamen und andererseits durch den Fensterstil vorgegeben. Beim Aufruf des Dialogfelds erzeugt die Dialogfeld-Verwaltung durch sukzessiven `CreateWindow`-Aufruf die spezifizierten Fenster. Nachdem Sie nun wissen, daß jedes Fenster einer Klasse angehört, sind Sie sicherlich neugierig, zu welcher Klasse denn ein Dialogfenster gehört. Nun, alle Dialogfenster gehören der Fensterklasse mit dem Namen »#32770« an. Die Fensterfunktion dieser Klasse geht beim Erhalten der `WM_CREATE`-Nachricht die Liste der Tochterfenster durch und legt diese an. Anschließend verwaltet sie die Adressierung der einzelnen Felder über die Tastatur (mit Tab-Taste, Pfeiltasten innerhalb von Gruppen usw.). Zusätzlich ruft sie bei den meisten Nachrichten, die sie erhält, die spezifizierte Dialogfunktion auf (*Bild 4*). In dieser kann der Programmierer benötigte Nachrichten abfangen und selbst auswerten. Wird ein von Null verschiedener Wert zurückgegeben, dann hat die Dialogfunktion die Nachricht selbst bearbeitet, und die Fensterfunktion ignoriert die Nachricht. Wird dagegen Null zurückgegeben, führt die Funktion ihre vorgesehene Aufgabe durch. Es werden leider nicht alle Nachrichten an die Dialogfunktion übergeben. Die Schnittstelle ist teilweise etwas unsauber, man kann jedoch im großen und ganzen damit leben. Auf weitere Probleme mit Dialogfeldern wird im zweiten Teil dieses Artikels eingegangen.

Veränderung von angelegten Fenstern und Fensterklassen

Oft müssen bereits angelegte Fenster oder sogar Fensterklassen in ihrem Aussehen oder ihrer Funktion nachträglich geändert werden. Die Fenstergröße und -position kann leicht mit den Funktionen `MoveWindow` oder `SetWindowPos` verändert werden. Ähnlich verhält es sich mit dem Fenstertext, der mit der Funktion `SetWindowText` neu eingestellt werden kann. Dieser Text muß übrigens nicht unbedingt wie bei einem Anwendungsfenster in der Titelleiste stehen, bei der Fensterklasse EDIT ist er zum Beispiel der einzeilig eingegebene Text.

Doch wie steht es mit den anderen in Tabelle 1 oder 2 angegebenen Parametern? Was tun, wenn man beispielsweise die Hintergrundfarbe oder den Darstellungsstil eines bereits angelegten Fensters ändern möchte, ohne es vorher zu vernichten und wieder erneut anzulegen? Windows erlaubt das Ändern der Fenster- und Fensterklassenparameter von bereits angelegten Klassen und Fenstern. Hierzu stehen die Funktionen `SetClassWord` und `SetClassLong` zum Ändern der Fensterklassen-Parameter und `SetWindowWord` und `SetWindowLong` zum Ändern der Fensterparameter zur Verfügung. Bei diesen Funktionen wird zunächst das betreffende Fenster angegeben, dann der in Tabelle 1 und 2 angegebene Index und schließlich der neue Wert. Die bisher gesetzten Werte von Fenster- und Fensterklassen-Parametern werden zurückgegeben oder lassen sich mit den entsprechenden Funktionen `GetClassWord` etc. ermitteln. Doch Vorsicht: Nicht alle Parametern dürfen geändert werden.

Die Veränderung bewirkt auch nur, daß der entsprechende Parameter im Speicher den neuen Wert erhält. Die Fensterdarstellung wird nicht automatisch angepaßt! Ist beispielsweise ein Fenster in Sinnbildgröße dargestellt und ändert man nun den Bezug auf ein anderes Sinnbild, dann tut sich zunächst am Bildschirm nichts. Erst wenn das Fenster erneut in Sinnbild-Darstellung gezeichnet wird, wird das neue Sinnbild angezeigt.

Unterklassen: Anpassung von vorhandenen Fensterklassen

Manchmal muß jedoch nicht die Form eines Fenster geändert werden, sondern dessen Verhalten. Was heißt dies konkret? Nun, Fenster werden über Nachrichten gesteuert, die von der Fensterfunktion ausgewertet werden. Wenn sich das Verhalten eines Fensters ändern, ändert sich also das Verhalten seiner Fensterfunktion bei bestimmten Nachrichten. Dies kann nur durch Änderung der Fensterfunktion gelöst werden. Wenn man aber nicht über den Quellcode der Fensterfunktion verfügt, beispielsweise weil die Fensterklasse vom System vordefiniert oder in einer Bibliothek abgelegt ist, bleibt nichts anderes übrig, als die Funktion komplett gegen eine neue auszuwechseln. Dann muß diese alle Aufgaben der alten Fensterfunktion miterledigen, also auch solche, die von der Verhaltensänderung gar nicht betroffen sind. Bei komplizierten Fensterklassen wie EDIT führt dies zu einem sehr großen Programmieraufwand, so groß, als hätte man die Fensterklasse komplett selbst definiert.

Viel einfacher ist dagegen folgender Weg: Die neue Fensterfunktion pickt sich die Nachrichten heraus, deren Reaktion geändert werden muß. Alle anderen Nachrichten werden dagegen wie bisher an die alte Fensterfunktion unverändert weitergeleitet. Die neue Funktion muß also nur einen kleinen Teil der Arbeit der alten Funktion erledigen. Durch diese Modifikation wurde die Klasse des angesprochenen Fensters nur teilweise geändert, bei Microsoft wird sie nun als Unterklasse (subclass) bezeichnet.

Programmtechnisch wird eine Unterklasse eines Fensters wie folgt realisiert: Nach dem Anlegen des Fensters wird seine bisherige Fensterfunktion mit `SetWindowLong` durch die neue ersetzt. Aus dieser wird dann mit Hilfe der Systemfunktion `CallWindowProc` die alte Fensterfunktion, die sogenannte Nachfolgerfunktion aufgerufen. *Bild 5* zeigt die Unterklassen-Technik im Schema. Es könnten auch mehrere Fensterfunktionen hintereinander angeordnet sein (Unter-Unter-Klassen). Wichtig ist das Grundprinzip, daß in einer Unterklassen-Fensterfunktion nur die Nachrichten geändert oder abgefangen werden dürfen, die tatsächlich bei der Verhaltensänderung des Fensters eine Rolle spielen. Alle anderen Nachrichten (und das sind die meisten) müssen unverändert an die Nachfolgerfunktion weitergeleitet werden.

Man muß sich Unterklassen-Fensterfunktionen so ähnlich vorstellen wie Vorzimmerdamen von leitenden Angestellten: Alle eingehenden Nachrichten (Telefon, Briefe usw.) werden vom Vorzimmer überprüft, ob sie den Chef interessieren, falls nicht, weggeworfen oder anderwertig bearbeitet. Unbekannte Nachrichten werden dagegen unverändert weitergeleitet. Manchmal muß die Dame für ihre Aufgaben zusätzlich Rücksprache mit dem Chef aufnehmen. Hierzu schickt sie selbst Nachrichten (Telefon, persönliches Gespräch) an ihn und er antwortet ihr direkt, ohne daß diese Kommunikation über das Vorzimmer hinausdringt.

Die Technik der Unterklassen zeigt, wie elegant die Kombination Nachricht-Fenster-Fensterfunktion ist. Würde ein Fenster durch viele verschiedene Funktionsaufrufe anstatt durch Nachrichten gesteuert werden, wäre die Unterklassentechnik nicht anwendbar.

Unterklassen eignen sich vor allem zur Verhaltensänderung vordefinierter Fensterklassen. In *[4]* wird auf Seite 410 bis 419 beispielsweise gezeigt, wie man die EDIT-Klasse dazu verwenden kann, Paßwörter oder andere geheime Information einzugeben. Die eingegebenen Zeichen sind im Eingabefeld nicht sichtbar, weil jedes eingegebene Zeichen in einem gesonderten Puffer gespeichert wird, und statt des Zeichens die Rücktaste an die eigentliche EDIT-Fensterfunktion geschickt wird, somit also das angezeigte Eingabefeld immer leer bleibt. Durch ähnliches Abfangen von bestimmten Zeichen kann man zum Beispiel auch erreichen, daß in einem Feld nur Buchstaben oder nur Zahlen oder nur gültige Geldwerte eingegeben werden können - wichtig beim Programmieren von Anwendungssoftware.

Im folgenden wird ebenfalls ein Beispiel für eine EDIT-Unterklasse vorgestellt, es ist jedoch etwas komplizierter als das eben zitierte. Es zeigt jedoch auch die Kommunikation der Unterklassenfunktion mit ihrer Nachfolgerin über lokale Nachrichtenströme.

Beispiel: Verbesserung der Fensterklasse EDIT

Wer schon einmal mit Microsofts EXCEL gearbeitet hat, hat sicherlich bemerkt, daß man bei ihm in jedem Fenster (auch innerhalb von Dialogfeldern) vom standardmäßigen Einfügen von Zeichen auf »Überschreiben« derselben durch Betätigen der [Einfg]-Taste hin- und herschalten kann. Das Überschreiben wird durch Anzeigen einer breiteren Einfügemarke kenntlich gemacht. Dies wurde einfach dadurch realisiert, daß das Zeichen, das überschrieben wird, ständig markiert ist (so als würde man ständig [Umsch][→] eingeben). Als weitere Erweiterung kann man mit [Strg] [Einfg] die markierten Zeichen in die Ablage kopieren. Diese beiden Möglichkeiten sind sehr praktisch, so daß eigentlich die Fensterklasse EDIT sie standardmäßig besitzen sollte. Leider wurden sie bei der Implementierung von EDIT in Windows weggelassen. Es bietet sich nun an, das Fehlen dieser beiden Fähigkeiten mit Hilfe einer Unter-

klasse von EDIT zu implementieren. Bei Excel wurde dies übrigens nicht so gemacht, sondern eine vollständig neue Fensterklasse namens XLFEDT erzeugt. Die Entwickler von Excel hatten es auch einfach, schließlich war der Quellcode von EDIT nur ein paar Türen weiter erhältlich!

Bei der eigentlichen Implementierung wurde mir schließlich klar, daß tatsächlich eine neue Fensterklasse sinnvoller ist als von EDIT eine Unterklasse zu erzeugen. Die Fensterklasse EDIT ist nämlich ziemlich schlampig implementiert und der Programmierer dachte wohl nicht daran, was einige Leute mit seiner Fensterfunktion vorhaben. Trotzdem, uns geht es ja ums Prinzip und das Ergebnis kann sich sehen lassen, denn beide Erweiterungen wurden prinzipiell implementiert.

Listing 1 zeigt das C-Listing des Programms SUBCLASS, die gehörige Ressourcen-Datei steht in *Listing 3*, die unvermeidbare Definitionsdatei in *Listing 4*. Zum Übersetzen wurde der Microsoft-C-Compiler, Version 5.1 verwendet. Es wird mindestens Version 5.0 des Compilers benötigt, da das Programm einige Features des neuen ANSI-C *[7]* besitzt. Das betrifft in erster Linie die neuartige Anordnung der Typen in den Funktionsargumenten, die etwas an Pascal erinnert. Die Linker-Anweisungsdatei ist in *Listing 5* aufgeführt. Auch hierzu ein Wort: Die Zeilen REALMODE und EXETYPE WINDOWS sind neu. Sie werden benötigt, wenn man mit dem OS/2-Linker LINK.EXE 5.01.21 (liegt MS-C 5.1 bei) Windows-Programme übersetzen will. Werden die Zeilen weggelassen, wird das Programm für den »protected modus« übersetzt und es geschehen beim Aufruf merkwürdige Dinge. Die beiden neuen Zeilen werden jedoch auch vom Vorgänger, dem Linker des Windows-SDK, LINK4.EXE 5.01.17 problemlos verarbeitet. Statt MOVEABLE kann jetzt der gebräuchlichere englische Begriff MOVABLE verwendet werden. Englisch ist halt auch für Amerikaner nicht ganz einfach. Alles wird übersetzt mit der MAKE-Datei in *Listing 2*.

Die Beschreibung der Anwendung kann ziemlich kurz ausfallen: Es werden zwei Fenster von der EDIT-Klasse angelegt: Das erstere belegt den gesamten Inhalt des Anwendungsfensters und erlaubt die Eingabe von mehreren Zeilen bis maximal 32 Kbyte. Genauso wurde übrigens auch das Programm NOTIZ.EXE implementiert, ergänzt um Datei-ein-/ ausgabe, Suchfunktion etc. Es handelt es sich hierbei um ein explizit mit `CreateWindow` angelegtes Tochterfenster des Anwendungsfensters. Man kann mehrere Zeilen eintippen, einen Bereich markieren, Text mit [Umsch][Einfg] aus der Ablage kopieren oder mit [Entf] bzw. [Umsch][Entf] den markierten Text löschen. Dies sind jedoch standardmäßige Merkmale der EDIT-Klasse, die beispielsweise auch bei der Eingabe eines Dateinamens in PAINT vorhanden sind. Neu ist, daß mit [Strg][Einfg] der markierte Text in die Zwischenablage kopiert wird, ohne daß er gelöscht wird. Neu auch der Überschreiben-Modus: Durch Betätigen der [Einfg]-Taste wird aus der Einfügemarke, sofern sie nicht am Zeilenende steht, wie bei EXCEL eine breite Einfügemarke, indem das aktuelle Zeichen markiert wird. Wird ein neues Zeichen eingegeben, überschreibt dies automatisch das markierte Zeichen. Anschließend wird das neue aktuelle Zeichen wieder markiert. Man kann weiterhin Text explizit markieren, einfügen, löschen etc., es wird jedoch ständig mindestens ein Zeichen markiert sein, außer die Einfügemarke ist am Zeilenende. Damit ist der Überschreiben-Modus realisiert. Er wird mit der [Einfg]-Taste ein- und ausgeschaltet.

Eine zweite Möglichkeit zum Testen ist ein Fenster der EDIT-Klasse innerhalb eines Dialogfelds. Dieses wird durch den Menübefehl »Zeile« aufgerufen. Es handelt sich hierbei um die Einzeilen-Version der EDIT-Klasse. Auch hier funktioniert die Kopierfunktion mit [Strg][Einfg]. Wegen eines Fehlers in der EDIT-Klasse ist jedoch der Überschreibmodus leider nicht möglich (siehe nächstes Kapitel).

Das Beispiel soll eigentlich nur zeigen, wie man Unterklassen realisiert und daß man sie sowohl bei mit `CreateWindow` angelegten Fenstern verwenden kann als auch innerhalb von Dialogfeldern.

Das C-Listing besitzt den schematischen Aufbau der meisten Windows-Programme, so daß hier nur die für die Unterklassentechnik relevanten Stellen beschrieben werden. Die Unterklassenfunktion wird genauso definiert wie jede andere Fensterfunktion. Im Beispiel besitzt sie den Namen `fwSubEdit`. Sie darf jedoch nicht direkt aufgerufen werden, sondern nur über den mit der bekannten Funktion `MakeProcInstance` erzeugten Funktionskopf. Dies wird in der Programm-Funktion `SetGlobalProcedure` durchgeführt, in der die Erzeugung der Funktionsköpfe zusammengefaßt ist. In `CreateAppWindow` wird dann zuerst das Anwendungsfenster und anschließend das Tochterfenster (`WS_CHILD`) mit der EDIT-Klasse angelegt. `IDEDIT` ist die Bezeichnung des Tochterfensters und mit `hwMain` ist der Bezug auf das Vaterfenster angegeben.

Am Ende von `CreateAppWindow` erfolgt die Einführung der Unterklasse, indem die Original-Fensterfunktion des EDIT-Fensters mit Hilfe der Funktion `SetWindowLong` durch die Unterklassen-Funktion ausgetauscht wird. Die Adresse der alten Funktion benötigen wir natürlich weiterhin und sie wird in der Variablen `rfwOrgEdit` gespeichert. Ab sofort werden die Nachrichten zur Funktion `rfwSubEdit` gesandt. Die Unterklasse bleibt erhalten, bis das EDIT-Fenster vernichtet wird. Dies wird erst beim Beenden der Anwendung stattfinden.

Ähnlich verhält es sich beim Einsatz der Unterklassentechnik in einem Dialogfeld, hierfür ist die Dialogfunktion `fdLine` zuständig. Ein Fenster kann natürlich nur dann eine Unterklasse erhalten, wenn es bereits angelegt ist. Letzteres geschieht aber hier erst durch den Aufruf der Funktion `DialogBox`, wenn das Dialogfeld dargestellt werden soll. Dies ist jedoch kein Problem: Nachdem `DialogBox` das Hauptfenster und die Tochterfenster des Dia-

logfelds angelegt (aber noch nicht angezeigt) hat, übersendet es an die Dialogfunktion `fdLine` die Nachricht `WM_INITDIALOG`. Dort kann dann wie bereits oben beschrieben, die Fensterfunktion ausgetauscht werden. Das war schon alles.

Wie man sieht, ist die Anwendung der Unterklassen-Technik sehr einfach. Man muß lediglich die Fensterfunktion eines angelegten Fensters durch die vorangestellte austauschen. Die eigentliche kreative Arbeit steht in der Unterklassen-Fensterfunktion bevor.

Die Unterklassen-Fensterfunktion

Die erste Nachricht, die es abzufangen gilt, ist `WM_SETFOCUS`. Da es nicht günstig ist, beim Aufruf eines Eingabefeldes sich gleich im Überschreiben-Modus zu befinden (zu leicht könnten Zeichen überschrieben werden), wird der Modus jedesmal abgeschaltet, wenn der Benutzer das Eingabefeld anwählt.

Die wichtigste Nachricht ist jedoch `WM_KEYDOWN`. Sie erscheint jedesmal, wenn eine Taste auf der Tastatur gedrückt wird oder beim Festhalten die automatische Tastenwiederholung einsetzt. Hierbei wird nicht der ANSI-Zeichencode übergeben, sondern der Wert der virtuellen Taste. Virtuelle Tasten umfassen auch alle Steuertasten wie [Einfg], [Alt], [F10] usw. Von diesen Tasten muß nur die [Einfg]-Taste (virtuellen Tastencode `VK_INSERT`) untersucht werden. Wurde sie zusammen mit der [Strg]-Taste gedrückt, so hat der Benutzer die Kopierfunktion aufgerufen. Deren Implementierung ist sehr einfach: Es muß einfach die Nachricht `WM_COPY` an die ursprüngliche Fensterfunktion geschickt werden. Dies funktioniert durch die Verwendung der Funktion `CallWindowProc`. Als Argument ist die Adresse der gesicherten alten Fensterfunktion angegeben.

Wurde die [Einfg]-Taste allein gedrückt, wird hiermit der Überschreiben-Modus ein- und ausgeschaltet. Der Überschreiben-Modus funktioniert nur bei einer EDIT-Klasse mit mehreren Zeilen (der Grund folgt unten). In der Unterklassen-Funktion wird daher zunächst der Fensterstil mit der Funktion `GetWindowLong` darauf überprüft.

Mit dem Betätigen der [Einfg]-Taste kann der Modus ein- oder ausgeschaltet werden. Bei letzterem muß auch die breitere Einfügemarke beseitigt werden. Dies darf natürlich nur geschehen, wenn die Markierung auch tatsächlich die Einfügemarke darstellt und genau ein Zeichen breit ist. In diesem Falle wird die Markierung auf null Zeichen verkürzt.

Alle anderen Nachrichten (`WM_KEYDOWN` mit anderen Tasten, `WM_PAINT`, `WM_SIZE` usw.) werden unverändert an die ursprüngliche Fensterfunktion weitergeleitet. Dies geschieht mit einem gemeinsamen `CallWindowProc`-Aufruf. Dessen Rückgabewert wird auch als Rückgabewert der Unterklassen-Funktion verwendet.

Sind alle Nachrichten bearbeitet, muß sichergestellt werden, daß während des Überschreiben-Modus das aktuelle Zeichen markiert bleibt. Denn bei jeder Zeicheneingabe, Maustasten-Betätigung usw. kann die Markierung gelöscht worden sein und muß daher erneut als breites Eingabezeichen gesetzt werden. Hierzu muß zunächst die Einfügeposition bestimmt werden. Das ist aber bei der EDIT-Klasse gar nicht möglich. Man kann nur den Bereich der markierten Zeichen erhalten. Da jedoch die Einfügemarke sich immer in diesem Bereich befindet, kann die Markierung zur Positionsbestimmung verwendet werden. Hierzu schickt die Unterklassen-Funktion eine `EM_GETSEL`-Nachricht an ihre Nachfolgerfunktion und diese liefert die Anfang- und Endposition der Markierung zurück. Sind beide Positionen gleich, ist kein Zeichen markiert, und es muß mit der `EM_SETSEL`-Nachricht das folgende Zeichen markiert werden. Sind dagegen mehrere Zeichen markiert, darf keine neue Selektion durchgeführt werden, denn in diesem Fall ist die Markierung vom Benutzer explizit über mehrere Zeichen ausgedehnt worden (beispielsweise zum Löschen von Zeichen).

Und nun zum oben angesprochenen Problem: Soll der Überschreiben-Modus richtig funktionieren, muß die blinkende Einfügemarke (Caret) links vom markierten Zeichen stehen. Bei dem Mehrzeilen-EDIT-Fenster funktioniert dies korrekt. Bei dem Einzeilen-EDIT-Fenster (verwendet im Dialogfeld) wird dagegen mit `EM_SETSEL` die Einfügemarke merkwürdigerweise hinter den markierten Bereich gestellt und dies läßt sich nicht korrigieren. Mit dem vorgestellten Prinzip läßt sich daher der Überschreiben-Modus beim Einzeilen-EDIT nicht realisieren. Die Realisierung ist deutlich komplizierter und würde den Rahmen des Artikels sprengen. Es wurde deshalb darauf verzichtet.

Michael Geary hat in *[5]* eine weitere interessante Anwendung von Unterklassen angesprochen: Die dynamische Änderung von Fenstereigenschaften während des Ablauf von Programmen. In seinem Beispiel verhalten sich in einem von ihm entwickelten Layout-Editor beispielsweise Schaltflächen (PushButton) bei der interaktiven Erzeugung eines Dialogfelds anders als bei deren späteren Benutzung, indem während der Erzeugungsphase die BUTTON-Klasse so abgeändert wird, daß beim Anklicken der Schaltfläche diese nicht aktiviert wird, wie man gewohnt ist, sondern markiert wird, um verschoben werden zu können.

Eine weitere sinnvolle Anwendung ergibt sich bei Lernprogrammen für eine Windows-Anwendung. Lernprogramme erlauben an einer Eingabestelle des zu erlernenden Programms oft nur eine kleine Auswahl an Eingabemöglichkeiten, damit der Auszubildende nicht durch fehlerhafte Eingaben versehentlich aus dem roten Faden des Lernprogramms aussteigen kann. Ein Lernprogramm sollte nicht ein Teil des zu erlernenden Programms sein, denn das würde die Komplexität für den Normalbetrieb unnötig erhöhen. Es muß deshalb von außen die Eingabemöglichkeiten kontrollieren können, etwa dadurch, daß es alle Nachrichten an das zu erlernende Programm abfängt und nur die gültigen weiterreicht, indem ein Systemeingriff (System Hook) *[6]* eingebaut wird. Das ist jedoch sehr aufwendig und fehlerträchtig, da das gesamte Verhalten einer Anwendung berücksichtigt werden muß. Eleganter ist es, der Fensterfunktion, die den Eingabefokus besitzt, im Lernmodus eine Unterklassen-Fensterfunktion voranzustellen, welche nur die gültigen Eingabenachrichten weiterleitet, die anderen dagegen blockiert.

Wie man sieht, kann man mit Unterklassen schöne Dinge schnell und einfach lösen (wenn wir einmal davon absehen, daß für das Beispiel die EDIT-Klasse nicht sauber genug implementiert wurde). Doch was macht man, wenn man ein komplett anderes Fenster braucht, zum Beispiel eine Laufwerk-Liste wie im MS-DOS-Fenster? Dann muß man eben seine eigenen Fensterklassen definieren. Das ist natürlich komplizierter als das »Aufblasen« einer vorhandenen Fensterklasse. Wie man Fensterklassen selbst erzeugt, können Sie in der Fortsetzung dieses Artikels im nächsten Heft erfahren, wenn zwei für Windows sehr ungewöhnliche Dialogelemente in einer ungewöhnlichen Anwendung vorgestellt werden. Marcellus Buchheit

[1] Microsoft Windows System Development-Kit, Version 2.0.
[2] Kevin P. Welch: Dynamischer Datenaustausch mit Windows-DDE. Microsoft System Journal, Januar/Februar 1988, Seite 22 bis 43.
[3] Petzold Charles: Programming WINDOWS, First Edition, 1988, Microsoft Press.
[4] Durant David, Carlson Geta, Yao Paul: Programmer's Guide to Windows, 2nd Edition, 1988, Sybex San Francisco.
[5] Craig Stinson: Die Datenbankoberfläche SQLWindows, Microsoft System Journal, September/Oktober 1988, Seite 4 bis 9.
[6] Marcellus Buchheit: Screen-Save-Funktion für MS-Windows, Microsoft System Journal, September/Oktober 1988, Seite 10 bis 25.
[7] Kernighan/Ritchie: The C Programming Language, Second edition 1988, Prentice Hall, Englewood Cliffs, New Jersey.
[8] Wirth, Nikolaus: Programming in Modula II, second edition 1983, Springer-Verlag, Berling-Heidelberg-New York.
[9] Stroustrup, Bjarne: The C++ Programming Language, Addison-Wesley 1986.

Ein Interview mit Ray Kanemori über die Zukunft von BASIC bei Microsoft:

Das neue QuickBASIC

Auf der diesjährigen CeBIT wird das neue QuickBASIC 4.5 in einer deutschen Version vorgestellt. QuickBASIC 4.5 hat zwar nur wenige Spracherweiterungen, bietet dafür aber ausgefeilte Lernhilfen und umfangreiche Hilfeinformationen. QuickBASIC bietet damit für den Einsteiger den gleichen Komfort, wie dies die Anwendungen von Microsoft schon lange tun. Wir haben uns mit Ray Kanemori, dem Product Marketing Manager für BASIC von Microsoft USA, über QuickBASIC 4.5 und die Zukunft von BASIC bei Microsoft unterhalten.

MSJ: In welche Richtungen wird sich Microsoft-BASIC und im besonderen QuickBASIC in den nächsten zwei Jahren entwickeln?

Kanemori: Wir werden mehr und mehr professionelle Sprachfeatures in BASIC haben. Eine der Richtungen, in die wir BASIC weiterentwickeln, sieht so aus, daß wir BASIC zu einer wirklichen Profi-Sprache machen wollen. Wir arbeiten zum Beispiel an verbesserter Leistung bei den vom Compiler erzeugten EXE-Dateien. In diesem Zusammenhang arbeiten wir auch an einer erweiterten Kapazität, so daß Sie auch in BASIC wirklich große Anwendungen schreiben können. Auch bei Overlays wird sich etwas tun. Weniger im QuickBASIC, das wir mehr als Einsteigersprache sehen, jedoch in unserem High-End-BASIC. QuickBASIC ist eine Sprache, die mehr auf lösungsorientierte Programmierer ausgerichtet ist, mit dem Ziel, daß sie sehr schnell produktiv werden können.

QuickBASIC ist eine sehr leistungsfähige Sprache und es gibt viele professionelle Entwickler, die QuickBASIC als ihr hauptsächliches Entwicklungswerkzeug verwenden. Man kann mit QuickBASIC sehr leistungsfähige Anwendungen schreiben, es gibt jedoch auch einige Einschränkungen, zum Beispiel in der Größe der Anwendungen, die man schreiben kann. Wir werden diese Einschränkungen entfernen.

Wir haben in letzter Zeit bemerkt, daß bei BASIC ein Trend zum High-End besteht. Die Benutzer haben gefordert, daß QuickBASIC eine größere Kapazität hat, so daß man größere Anwendungen entwickeln kann. Viele Leute meinen, daß die ausführbaren Dateien zu groß sind. Wir werden bei der Größe und auch der Geschwindigkeit der erzeugten Programme einige Optimierungen vornehmen. Es spricht nichts dagegen, daß BASIC in Punkto Geschwindigkeit langsamer sein sollte als C oder Pascal.

MSJ: Heißt das, daß Sie den Optimierer aus C und Fortran in BASIC einbauen werden?

In BASIC werden auch schon derzeit Optimierungen vorgenommen und es ist eigentlich schon recht schnell. Es wird aber sicherlich einiges von dieser Technologie übernommen werden.

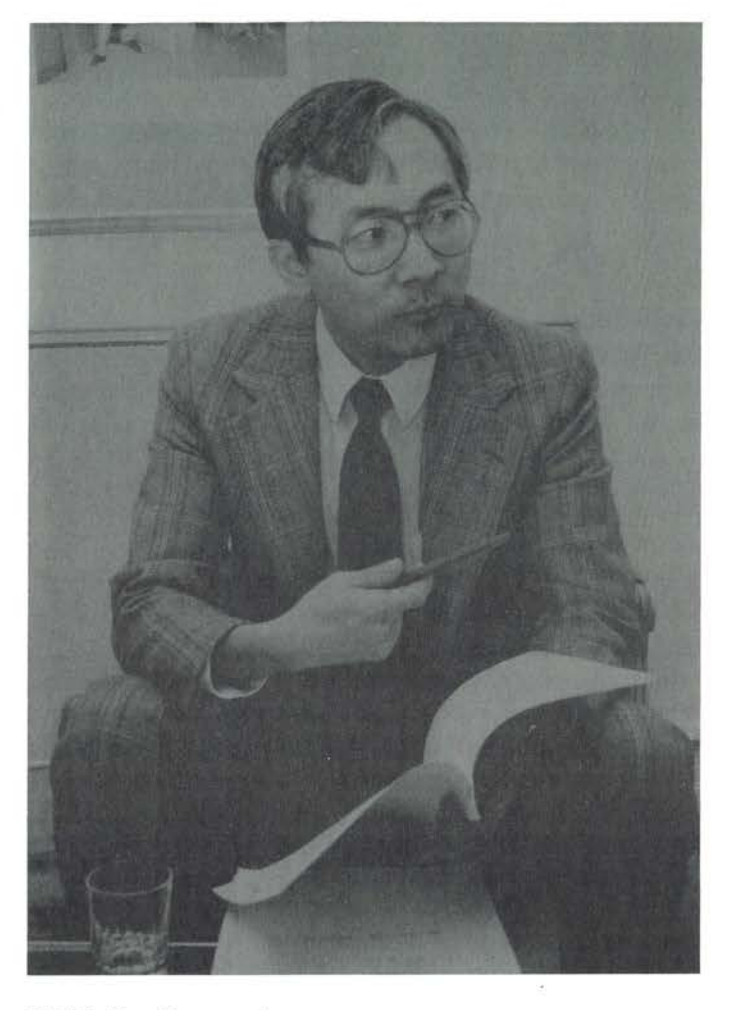


Bild 1: *Ray Kanemori*

MSJ: BASIC wird also stärker auf den Profi ausgerichtet?

Kanemori: Nicht QuickBASIC 4.5. In der Version 4.5 haben wir uns darauf konzentriert, viele Lernhilfen einzubauen und Programmierern dabei zu helfen, schnell produktiv zu werden. QuickBASIC 4.0 war ein großer technologischer Fortschritt. Mit seiner Threaded-P-Code-Technologie kann man Anwendungen sehr schnell entwickeln. Wir haben in 4.0 auch viele Neuheiten wie Records und Rekursion eingeführt, die QuickBASIC sehr leistungfähig gemacht haben.

Die Sprache wurde in der Version 4.5 nur bei drei oder vier Anweisungen erweitert, und zwar wurde der Bereich der Ereignisabfrage (ON EVENT und benutzererdefinierte Ereignisse) um die Möglichkeiten von BASIC 6.0 erweitert. Es ist so, daß wir QuickBASIC auch weiter als das Produkt zum Erlernen des Programmierens sehen: ein Weg, die Sprache BASIC sehr schnell zu meistern und sehr schnell produktiv zu werden. BASIC 6.0 und seine Nachfolgeprodukte sehen wir als Sprachen für das obere Marktsegment der professionellen Programmierer. Viele professionelle Programmierer verwenden BASIC, wir werden deshalb in den Folgeprodukten von BASIC 6.0 mehr professionelle Möglichkeiten bieten, die BASIC für Profis attraktiver machen.

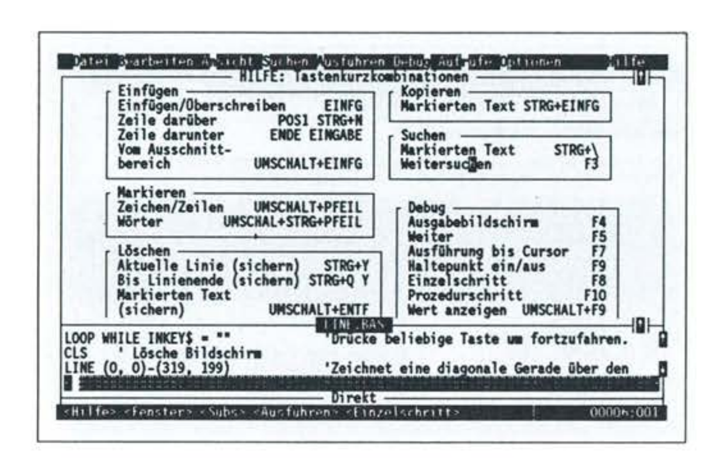


Bild 2: *Die wichtigste Neuheit von QuickBASIC 4.5 ist der Ratgeber, der das komplette QuickBASIC-Handbuch mit vielen Beispielen enthält. Es können nicht nur Informationen zur Bedienung...*

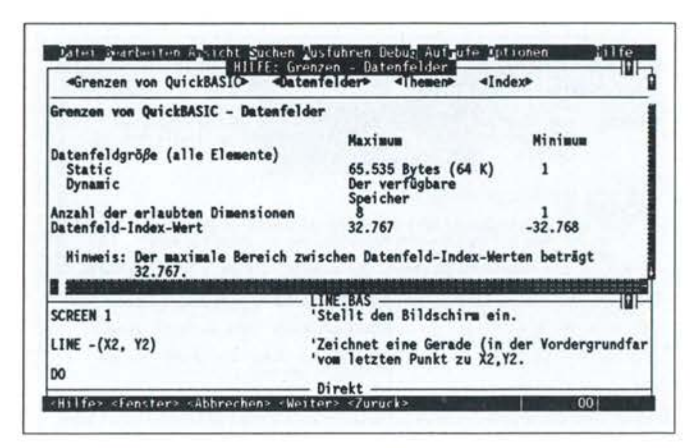


Bild 3: *... sondern auch alle anderen Informationen, die gewöhnlich in einem Handbuch stehen, abgerufen werden. Zahlreiche Verweise erleichtern das Auffinden verwandter Themen.*

MSJ: Können Sie etwas detaillierter auf einzelne Features oder Befehle eingehen?

Kanemori: Dazu wäre es noch zu früh, doch wie ich schon gesagt habe, wird es dabei im wesentlichen um die Steigerung der Produktivität und die Größe der Anwendungen gehen. Es werden sehr viele geschäftliche Anwendungen mit BASIC entwickelt, die Erweiterungen haben darum dieses Einsatzgebiet zum Ziel, daß man also geschäftliche Anwendungen mit BASIC noch besser schreiben kann, zum Beispiel neue mathematische Bibliotheken oder Matrix-Anweisungen.

Ferner ist für das High-End-Produkt die Programmer's Workbench geplant, eine integrierte Programmierumgebung ähnlich wie die Quick-Umgebung, aber mit der Möglichkeit, daß man seine eigenen Tools in diese Umgebung integrieren kann, zum Beispiel CodeView. Man kann also bei der Entwicklung in der gewohnten Umgebung bleiben und dennoch auch andere Tools einsetzen.

MSJ: Wie sieht es mit den noch bestehenden Einschränkungen von BASIC aus? Ich selbst haben zum Beispiel bei der Programmierung mit QuickBASIC häufig Probleme mit der Beschränkung des Stringspeichers auf 64 Kbyte.

Kanemori: Wir werden bei unserem High-End-Produkt in dieser Richtung sicher etwas unternehmen, man wird irgendwann den ganzen freien Speicher für Strings verwenden können.

MSJ: Die Handbücher zu QuickBASIC 4.5 sind nicht mehr so umfangreich, wie bei früheren Versionen. Warum?

Kanemori: Wir haben festgestellt, daß die meisten Benutzer nicht ins Handbuch schauen, sondern Informationen aus den Hilfebildschirmen abrufen und gerne dort mehr Informationen sehen würden. Wir glauben deshalb, daß der Trend zu mehr Online-Informationen geht. Unser Quick-BASIC-Ratgeber ist ein ausgezeichnetes Beispiel für diese Technologie (*Bilder 2 bis 5*).

Man kann Beispielprogramme aus dem Ratgeber heraus in ein eigenes Programm hinein kopieren und ausführen. Wenn man zu einer Anweisung detailliertere Informationen möchte, braucht man nur auf dem Bildschirm das Wort »Detail« anklicken und erhält diese Informationen. Ich glaube, daß dies für den Anwender nützlicher ist, als Handbücher. Es gibt alle Referenzinformationen und Beispiele direkt im Produkt und ich denke, daß man dies beim Programmieren lieber alles Online als in einem Handbuch hat. Natürlich gibt es auch noch die normalen Handbücher inklusive Referenzinformationen, doch ist alles knapper gehalten und auch die Beispiele sind in den Online-Ratgeber verlagert worden. So kann der Anwender alle diese Beispiele durch Kopieren auch direkt bei der Programmierung nutzen, ohne sie umständlich selbst eingeben zu müssen.

Wer die bisherige Form vorzieht, kann diese Handbücher auch weiterhin optional von Microsoft erhalten. Doch wir haben die Erfahrung gemacht, daß den meisten Benutzern eine kurze Referenz auf Papier reicht.

MSJ: Was hat sich sonst noch bei QuickBASIC 4.5 getan?

Kanemori: Wir haben zahlreiche Untersuchungen mit Anwendern durchgeführt, weil wir wissen wollten, wie sie vorgehen, um mit QuickBASIC möglichst schnell möglichst produktiv zu werden. Dabei zeigte sich, daß die Menüpunkte von QuickBASIC 4.0 von Anwendern als zu umfangreich angesehen wurden. Wir haben das dann genauer untersucht und versucht herauszufinden, was für den leichten Einstieg die optimale Anzahl an Menüpunkten ist. Daraus entstand die Kurzform der Menüs in QuickBASIC 4.5, bei denen es sich um eine Untermenge der vollständigen Menüs handelt, und zwar um die unbedingt notwendigen und die am häufigsten benötigten. Dadurch kann der Anwender die Lernphase um einiges verkürzen. Das ist also ähnlich wie bei Microsoft Excel.



Bild 4: *QuickBASIC erlaubt im Hilfefenster viele Befehle aus dem Editor. So kann man im Hilfetext zum Beispiel wie im Programm nach Wörtern und Begriffen suchen.*



Bild 5: *Wenn man ein für das eigene Programm nützliches Programmbeispiel oder Modul findet, kann man es markieren und in das eigene Programm hineinkopieren.*

Auch das neue Installationsprogramm von QuickBASIC zielt darauf ab, dem Anwender den Start zu erleichtern und ihn möglichst schnell zum Arbeiten kommen zu lassen. Ebenso das Lernprogramm für QuickBASIC, mit dem man sich in 10 Minuten mit dem Umgang mit QuickBASIC 4.5 vertraut machen und die Umgebung kennenlernen kann. Doch am wichtigsten war uns der QuickBASIC-Ratgeber. Wir haben sehr viel Arbeit in diese Technologie gesteckt und versucht, wesentlich mehr Informationen als bisher in die Programmierumgebung hineinzubekommen.

Aber auch im Handbuch hat sich etwas getan. So befindet sich darin jetzt ein Einführungskurs mit einem umfangreichen Beispiel für eine Dateiverwaltung, dessen Entwicklung dem Benutzer Schritt für Schritt verdeutlicht wird.

Auch beim Programmieren wird der Benutzer nun besser informiert. So wurden die Fehlermeldungen stark verbessert und enthalten jetzt genaue Fehlerbeschreibungen und Tips für die Korrektur. Zu den kleinen, aber ungemein nützlichen Erweiterungen gehört, daß man sich nun jederzeit den aktuellen Wert einer Variablen anzeigen lassen kann, indem man auf diese Variable zeigt und eine Funktionstaste drückt.

MSJ: Wird es in Zukunft auch BASIC-Versionen für andere Umgebungen geben, zum Beispiel für Windows oder den Presentation Manager?

Kanemori: Wir sind derzeit dabei, uns andere Umgebungen unter diesem Gesichtspunkt anzusehen. Genaueres kann ich dazu momentan noch nicht sagen, doch sehen Sie sich einmal den neuen QuickBASIC-Compiler für den Macintosh an. Hier wird die grafische Umgebung voll unterstützt und es werden auch Möglichkeiten für den Zugriff auf die Toolbox-Funktionen geboten. Dieser Compiler ist ein gutes Beispiel dafür, wie Microsoft BASIC in einer grafischen Fensterumgebung aussieht und arbeitet. Er zeigt, daß man mit BASIC auch in diesen Umgebungen einfacher programmieren kann, als mit Programmiersprachen wie C.

MSJ: Wie sehen Ihre Pläne für GW-BASIC aus?

Kanemori: Wir werden GW-BASIC weiter mit MS-DOS ausliefern und auch weiter pflegen, weitere Pläne bestehen derzeit nicht.

MSJ: In der Presse war in den letzten Wochen gelegentlich etwas von einem Embedded BASIC zu hören. Worum handelt es sich dabei und was hat es mit BASIC zu tun?

Kanemori: Embedded-BASIC ist ein langfristige Entwicklungsrichtung für die Integration unserer Anwendungsprogramme. Die Sprache wird QuickBASIC sehr ähnlich sein; viele Dinge der QuickBASIC-Technologie werden übernommen werden.

MSJ: Herr Kanemori, vielen Dank für dieses Gespräch.

Eigenschaft	2.0	3.0	4.0	4.5
On-Line QB-Ratgeber	Nein	Nein	Nein	Ja
Funktion der rechten Maustaste wählbar	Nein	Nein	Nein	Ja
Stoppbedingungen Variablen/Ausdrücke	Nein	Nein	Nein	Ja
Setzen von Standard-Suchpfaden	Nein	Nein	Nein	Ja
Benutzerdefinierte Typen	Nein	Nein	Ja	Ja
IEEE-Format, math. Koprozessor	Nein	Ja	Ja	Ja
On-Line-Hilfe	Nein	Nein	Ja	Ja
Lange (32-bit) Ganzzahlen	Nein	Nein	Ja	Ja
Zeichenketten fester Länge	Nein	Nein	Ja	Ja
Syntaxprüfung bei der Eingabe	Nein	Nein	Ja	Ja
Binär-Datei-E/A	Nein	Nein	Ja	Ja
FUNCTION-Prozeduren	Nein	Nein	Ja	Ja
CodeView-Unterstützung	Nein	Nein	Ja	Ja
Kompatibilität zu anderen Sprachen	Nein	Nein	Ja	Ja
Mehrere Module im Speicher	Nein	Nein	Ja	Ja
Kompatibel zu residenten Programmen	Nein	Ja	Ja	Ja
Einfüge-/Überschreib-Modus	Nein	Ja	Ja	Ja
Tastaturschnittstelle im WordStar-Stil	Nein	Nein	Ja	Ja
Rekursion	Nein	Nein	Ja	Ja
Fehler-Listings bei separater Kompilierung	Nein	Ja	Ja	Ja
Assembler-Listings bei sep. Kompilierung	Nein	Ja	Ja	Ja

Tabelle 1: *Eine Gegenüberstellung der Fähigkeiten der QuickBASIC-Versionen von 2.0 bis 4.5 zeigt, welche Fortschritte die Sprache bei Microsoft in den letzten Jahren gemacht hat.*

Termine ... Termine ... Termine ... Termine

Mit Microsoft-Seminaren sicher in die Zukunft

Die Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung unter OS/2 und Windows nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Die Dozenten befassen sich auch im persönlichen Gespräch ausführlich mit den individuellen Forderungen und Problemen der Teilnehmer. So bekommen professionelle Entwickler durch professionelle Schulung die Möglichkeit, ihren hohen Wissensstand den neuen Gegebenheiten anzupassen.

Jeder Interessent in der Bundesrepublik Deutschland, der Schweiz und Österreich hat so die Chance, sich mit der neuen Welt von Microsoft OS/2 und Microsoft Windows auseinanderzusetzen.

Das Microsoft OS/2 Einführungsseminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft OS/2 kennen und erhalten einen Überblick über die Fähigkeiten und Programmierschnittstellen dieses Betriebssystems. Während des Seminars haben die Teilnehmer die Möglichkeit, das Gelernte anhand von Übungsaufgaben für sich selbst zu überprüfen.

Ort	Datum	Tag
Düsseldorf	19./20.06.89	Mo./Di.
München	29./30.05.89	Mo./Di.

Der Microsoft OS/2 Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Family-API-Programme zu schreiben und Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen; außerdem können sie die erweiterten Speicherverwaltungsmöglichkeiten des Intel 80286 nutzen und mit Hilfe des MS-OS/2 Memory Managers programmieren. Dieses Seminar ist übrigens nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	21./22./23.06.89	Mi./Do./Fr.
München	31.05./1./2.06.89	Mi./Do./Fr.

Das Microsoft Windows Einführungsseminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft Windows kennen und erhalten einen Überblick über dessen Fähigkeiten und Programmierschnittstellen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	26./27.06.89	Mo./Di.
München	05./06.06.89	Mo./Di.

Der Microsoft Windows Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Benutzerschnittstellen zu erstellen, die grafische Programmierschnittstelle zu nutzen, die Routinen zum Memory Management anzuwenden und dynamische Bibliotheken zu erstellen und zu benutzen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	28./29./30.06.89	Mi./Do./Fr.
München	07./08./09.06.89	Mi./Do./Fr.

Microsoft Presentation Manager Einführungsseminar

Dieses Seminar erleichtert das Umsteigen von Microsoft Windows auf den Microsoft OS/2 Presentation Manager.

Der Teilnehmer lernt die Unterschiede zwischen Microsoft Windows und dem Microsoft OS/2 Presentation Manager kennen und lernt, wie er seine Windows-Applikationen auf den PM übertragen kann.

Ort	Datum	Tag
Düsseldorf	19.05.89	Fr.
München	12.06.89	Di.

Microsoft LAN Manager für Windows-Programmierer

Ort	Datum	Tag
Düsseldorf	16.05.89	Fr.
München	13.06.89	Di.

Mitteilungen .... Mitteilungen .... Mitteilungen

Common-X-Interface: Brücke zwischen Unix und PC-Umgebungen

Hewlett-Packard und Microsoft haben kürzlich mit dem Common-X-Interface (CXI) ein grafisches Bediener-Interface für Unix-Betriebssysteme vorgestellt. Die Bedienungsoberflächen von Unix-Systemen erhalten damit das gleiche zweidimensionale Erscheinungsbild wie MS-DOS mit Microsoft Windows und MS OS/2 mit dem Presentation Manager. Darüber hinaus stellte Hewlett-Packard eine separate CXI-Version mit dreidimensionalem Erscheinungsbild vor.

CXI stellt für Softwareentwickler Richtlinien und Werkzeuge zur Entwicklung von Anwendungen mit konsistentem Bediener-Interface zur Verfügung. Die Gleichartigkeit ermöglicht es Anwendern, die mit Microsoft Windows oder dem Presentation Manager vertraut sind, durch CXI ohne großen Lernaufwand mit jedem Unix-System zu arbeiten.

CXI basiert auf dem X-Windows-System, einem etablierten Standard in der Unix-Welt. Es wurde von Hewlett-Packard und Microsoft entwickelt. Dies war eine Reaktion auf die Forderung der Open Software Foundation nach Standardisierung der Bedieneroberfläche von Unix.

Die Brücke zwischen Unix und PC

Mit dem Common-X-Interface haben Hewlett-Packard und Microsoft ein zuverlässiges Bediener-Interface für Unix-Systeme als Brücke zwischen der PC- und der Unix-Systemwelt geschaffen. Die Attraktivität des CXI ergibt sich aus zwei Gesichtspunkten: Konsistenz mit der Umgebung von Microsoft Windows, die Millionen von Computer-Anwendern vertraut ist - und sofortige Verfügbarkeit. Durch CXI sind Hardware- und Systemanbieter in der Lage, ihren Anwendern eine einheitliche Produktlinie über die drei Betriebssystem-Umgebungen MS-DOS, MS OS/2 und Unix zu bieten. Unternehmen, die MS-DOS, MS OS/2 und Unix-Systeme unter einem Dach einsetzen, werden feststellen, daß die Einarbeitungszeit der Anwender erheblich verringert wird, weil sie sich nur noch mit einer Bedienungs-Oberfläche vertraut machen müssen.

Die CXI-Bestandteile

CXI besteht aus drei Grundelementen:

- einem »Style Guide«, der ein zum MS OS/2 Presentation Manager konsistentes Verhalten dokumentiert,
- der HP-X-Widget Anwendungs-Programmier-Interface-Spezifikation und -Software (API) sowie
- dem HP-Window-Manager, einer Implementation der MS OS/2 Presentation Manager-Funktionen.

Der »Style Guide« bietet flexible Richtlinien bei der Bildschirmdarstellung des Bediener-Interface. Softwareentwickler können auf der Basis dieser Richtlinien eigene Vorstellungen bei der Gestaltung ihrer Anwendungs-Bedienungsoberflächen realisieren. Die Konsistenz in Layout und Verhalten des Bediener-Interfaces wird dabei trotzdem gewahrt.

Der HP-Window-Manager, der gemäß dem CXI-»Style Guide« ausgelegt ist, steuert Bewegung, Position und Format von Fenstern auf dem Bildschirm.

Hewlett-Packard und Microsoft gaben gleichzeitig mit der Vorstellung des CXI bekannt, daß sie gemeinsam den Presentation Manager/X für Unix-Systeme als Erweiterung von CXI entwickeln. Hewlett-Packard plant außerdem die Entwicklung einer HP-NewWave Software-Anwendungsumgebung für Unix-Systeme unter Nutzung von CXI als grafisches Bediener-Interface.

Verfügbarkeit

Der CXI-Source-Code ist in hohem Grade portabel und wird durch Technologie-Lizenzvereinbarungen auf breiter Basis verfügbar sein. Der CXI-Style Guide und die API-Software sind für Softwareentwickler und Systemhersteller sofort lieferbar. Der Window Manager, den Anwender benötigen, um Anwendungen laufen zu lassen, wird im zweiten Quartal 1989 zur Verfügung stehen.

CXI und Presentation Manager/X

Mit Common-X-Interface (CXI) und Presentation Manager/X trägt Microsoft der Problematik der Standardisierung im Software-Bereich Rechnung. Die heutige Situation im Computermarkt ist gekennzeichnet von dem Bemühen, Anwendern den Komfort einer einheitlichen Bedieneroberfläche zu bieten. So ist zum Beispiel bei der Open Software Foundation (OSF) und der europäischen X/Open-Gruppe die Festlegung eines Standards das vorrangige Ziel.

Common-X-Interface und Presentation Manager/X wurden in gegenseitiger Abstimmung entwickelt und ergänzen sich daher. Sie schlagen die Brücke von MS-DOS und MS OS/2 zu Unix-Systemen und bieten folgende Vorteile:

1. CXI und PM/X versetzen Software-Hersteller und OEMs in die Lage, eine einheitliche Produktlinie für MS-DOS, MS OS/2 und Unix-Betriebssysteme bereitzustellen.
2. Unix-Anwendungsprogramme können so ausgelegt werden, daß sie die gleiche Bedienungsoberfläche und Handhabung aufweisen wie Anwendungsprogramme unter Microsoft Windows/Presentation Manager. Die Anwender werden also größtenteils bereits mit dem Bedienungs-Interface vertraut sein, wenn sie Unix-Anwendungen einsetzen wollen, die unter CXI laufen.
3. OEMs und Endanwender können darauf vertrauen, daß die Integration von Unix-Systemen in bereits bestehende Umgebungen reibungslos verläuft, weil das CXI und das PM/X-Interface auf der Mehrheit der schon installierten Systeme konsistent sein werden.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

4. CXI und der PM/X basieren auf etablierten Standards der Unix-Welt, wie zum Beispiel der C-Programmiersprache und X-Windows. Entwickler von Unix-Anwendungen werden beim Umstieg auf CXI und PM/X also weiterhin mit den vertrauten und erprobten Unix-Werkzeugen arbeiten können.

Warum ist ein Standard bei der Benutzeroberfläche von so großer Bedeutung? Die Arbeit mit dem Computer soll für den Anwender immer einfacher werden. Dazu gehört, daß er sich nicht permanent mit neuen Befehlsstrukturen, Funktionen und einem fremden Bildschirmgesicht auseinandersetzen muß. Im Grunde soll er mit nur einer Bedieneroberfläche auf allen Computern arbeiten können. Das bringt zwei große Vorteile:

- Statt zeitraubender und lästiger Einarbeitung sofort effektives Arbeiten.
- Die einfache Handhabung erhöht die Akzeptanz des Computers generell.

Eine einheitliche Benutzeroberfläche ist nicht nur für den Anwender von Vorteil. Entwickler von Anwendungen mußten sich bisher entscheiden, für welche grafische Benutzeroberfläche sie ihre Programme schreiben. Die Zielgruppe war begrenzt. Die Herstellung von Anwendungen für eine bestimmte Oberfläche ist aber relativ teuer und eine Portierung auf andere Umgebungen verursacht hohe Kosten. Erweitert sich nun die Zielgruppe durch eine standardisierte grafische Benutzeroberfläche auf unterschiedlichen Betriebssystemen, ist dies für Software-Hersteller eine Möglichkeit, größere Stückzahlen abzusetzen. Damit verteilen sich die Entwicklungskosten entsprechend, die Anwendungsprogramme werden billiger.

Bislang haben nur wenige Software-Hersteller mit ihren Anwendungen erfolgreich verschiedene Betriebssysteme unterstützt. Microsoft ist mit seinen Anwendungen für MS-DOS, MS OS/2, Xenix und das Macintosh-Betriebssystem eine Ausnahme.

Microsoft Windows: Ein Standard entsteht

Microsoft kündigte 1984 Windows an, eine grafische Bildschirmoberfläche für MS-DOS-PCs. Jeden Monat werden inzwischen mehr als 50.000 Einheiten Windows ausgeliefert. Die Software wird sowohl über den Handel vertrieben als auch zusammen mit Hardware ausgeliefert. Installiert ist Windows bereits auf Rechnern der Firmen Hewlett-Packard, Zenith, IBM, Compaq, Wang, Unisys und NCR. Ein weiterer Beweis für die hohe Akzeptanz des Produkts im Markt ist sein Einsatz bei großen internationalen Unternehmen wie zum Beispiel Covia/United Airlines, Arthur Anderson, Manufacturers Hanover Trust und Merill Lynch. Bis Ende des Jahres 1989 werden voraussichtlich 2,5 Millionen Microsoft Windows Einheiten installiert sein. Damit ist Windows die am meisten verbreitete Benutzeroberfläche und der Grafik-Interface-Standard in der MS-DOS-Welt.

Daß MS-DOS-Systeme im Bereich der Tischcomputer - dem größten Segment im Computermarkt - eine vorherrschende Position haben, stellen auch jene Computer-Hersteller fest, die Unix-Systeme an Großkunden verkaufen wollen. Sie finden dort eine breite Basis an MS-DOS/Windows-Peresonalcomputer und MS OS/2-Systemen vor, in die der Unternehmer viel investiert hat. Er hatte Ausgaben in den Bereichen

- Computer-System-Training,
- Anwendungs-Training,
- Dokumentation,
- Trainingshilfen auf Computer-Basis,
- Programme.

Diese Investitionen kann der Unternehmer nur dann schützen, wenn neue Anwendungen den Anwender nicht in den Stand des Computerlaien zurückwerfen. Eine durchgängige Produktlinie ist daher ebenso erforderlich wie ein Bedienungsstandard. Ob Unix, MS-DOS oder MS OS/2 - gefragt ist eine Bedieneroberfläche, die für alle drei Betriebssysteme gilt. Bislang gibt es nur viele Standardisierungsbemühungen, im Unix-Bereich existiert kein einziger eindeutiger Standard für ein grafisches High-Level-Bediener-Interface. Das X-Windows-System bietet zwar eine Standardisierung auf niederer Ebene an, im High-Level Bereich klaffte aber bisher eine Lücke. Gefüllt wird sie durch das Common-X-Interface und den Presentation Manager/X von Microsoft.

Das Microsoft Betriebssystem OS/2 repräsentiert die zweite Generation von PC-Betriebssystemen für den kommerziellen Einsatz und erweitert die in MS-DOS zur Verfügung stehenden Möglichkeiten. MS OS/2 umfaßt bereits eine integrierte Unterstützung für eine grafische Benutzeroberfläche ähnlich Windows. Diese Erweiterung des Betriebssystems ist der Presentation Manager. Er basiert auf den gleichen Spezifikationen wie Windows. Anwender von Windows können daher auf die leistungsstarke MS OS/2-Betriebssystem-Umgebung ohne Einarbeitung umsteigen.

Hinsichtlich der führenden Anwendungspakete für die Büroautomation wird der Presentation Manager 1989 die von unabhängigen Software-Häusern am breitesten unterstützte grafische Bedieneroberfläche sein. Alle führenden Software-Hersteller, einschließlich Lotus, Ashton Tate und Microsoft selbst, haben Software-Produkte für den Presentation Manager angekündigt oder bereits vorgestellt.

Microsoft und Hewlett-Packard haben sich die Aufgabe gestellt, eine Netzwerk-Software zu entwickeln, mit der MS-DOS, MS OS/2 und Unix-Systeme in einem lokalen Netzwerk zusammenarbeiten können. In Anlehnung an den LAN Manager des MS OS/2-Betriebssystems wird dieses Netzwerkprodukt für Unix den Namen LAN Manager/X tragen.

Mitteilungen .... Mitteilungen .... Mitteilungen

Zusammenwirken von MS OS/2 LAN Manager und IBM OS/2 LAN Server demonstriert

Auf der COMDEX-Trade-Show in Las Vegas, Nevada, hat Microsoft das direkte Zusammenwirken zwischen dem Microsoft OS/2 LAN Manager und dem IBM OS/2 LAN Server beeindruckend in der Praxis demonstriert. Microsoft führte vor, wie die Anwender LAN Manager-Produkte, Workstations, Server, MS-DOS-Computer und MS OS/2-Systeme nach ihren speziellen Anforderungen in Verbindung mit dem IBM OS/2 LAN Server einsetzen können.

Der MS OS/2 LAN Manager und der IBM OS/2 LAN Server basieren auf einheitlichen Standard-Interfaces wie dem SMB-Netzwerk-Protokoll, dem NetBIOS-Interface sowie dem NetBEUI/DLC-Netzwerk-Transport-Stack. Diese Standardschnittstellen ermöglichen es, daß LAN Manager und LAN Server ohne Gateways und Protokollkonverter direkt zusammenarbeiten können. Wie Christian Wedell, Geschäftsführer der Microsoft GmbH in München-Aschheim, unterstrich, »wird das hohe Niveau der direkten Interoperabilität einen bedeutenden Einfluß auf das Wachstum und die Struktur des PC-Netzwerkmarktes haben. Dieses Niveau kann mit herstellerspezifischen Netzwerk-Produkten auf der Basis von nichtstandardisierten Schnittstellen und Protokollen nicht erreicht werden.«

Die Funktionen und Schnittstellen, die die Zusammenarbeit zwischen dem LAN Manager und dem LAN Server ermöglichen, sind integrierter Bestandteil des Standard LAN Managers. Damit können alle OEM-Anwender des LAN Managers ihren Kunden das Zusammenwirken von IBM OS/2 LAN Server und Microsoft OS/2 LAN Manager bieten. Die Firmen 3Com Corporation und Torus Systems Ltd, die den LAN Manager als erste ausliefern, wollen diese Konfiguration von LAN Manager und LAN Server forciert unterstützen. Eric Benhamou, Vice President und General Manager der 3Com »Software Products Division«, sieht gegenüber Netzwerk-Lösungen anderer Anbieter einen erheblichen Vorteil darin, daß der von seinem Unternehmen angebotene 3+Open LAN Manager mit dem IBM OS/2 LAN Server direkt zusammenarbeiten kann. Stephen Ives, Managing Director von Torus, ergänzt, daß sein Unternehmen eng mit den Wiederverkäufern und Kunden kooperieren möchte, um so sicherzustellen, daß der LAN Manager gemeinsam mit dem IBM OS/2 LAN Server eingesetzt werden kann.

Die Demonstration des »Zusammenspiels« zwischen LAN Manager und LAN Server war nur ein Element der LAN Manager Expo, die im Rahmen der Las-Vegas-COMDEX stattfand. Mehr als ein Dutzend OEM-Firmen zeigte unter Einsatz des LAN Managers die Vernetzung von über 60 Computern einschließlich PCs unter MS OS/2 und MS-DOS, Apple Macintosh Computer, Sun-Workstations, DEC VAX Computer und Unix-Systeme. Die eingesetzten Netzwerk-Medien umfaßten EtherNet, Token Ring, Arcnet und Starlan. Die mehr als 60 Maschinen und vier verschiedenen Netzwerk-Medien waren über eine Reihe von kommerziell verfügbaren Netzwerk-Programmen wie IBM-»Token Ring Source Routing Bridge« und »Ungermann Bass Net/One Token Ring/EtherNet Data Link Bridge« verbunden und bildeten ein heterogenes LAN Manager Netzwerk.

Des weiteren zeigten mehr als ein Dutzend unabhängiger Software-Hersteller als spezielle Demonstration der Interoperabilität zwischen LAN Server und LAN Manager Anwendungen, die transparent in der LAN Manager-/LAN Server-Umgebung liefen. Unter diesen Anwendungen stachen einige Frontend-Anwendungen für den Ashton-Tate/Microsoft SQL Server und den SQL Server selbst hervor.

Um sicherzustellen, daß Systeme, die durch LAN Manager-OEMs unterstützt werden, ohne weiteres mit dem IBM OS/2 LAN Server zusammenarbeiten, hat Microsoft ein »Conformance Certification Program« für LAN Manager OEM-Kunden entwickelt. Dieses Programm wird von »DWB Associates of Beaverton«, Oregon, durchgeführt. Dabei erhalten OEM-Hardwaretreiber nach einer entsprechenden Prüfung ein Zertifikat, das die Interoperabilität und Konformität zu Microsofts »Media Access Control«-Treiber-Interface bescheinigt.

Bildschirmhersteller unterstützen den Microsoft OS/2 Presentation Manager

Namhafte unabhängige Hersteller von Bildschirm-Terminals und Video-Treibersoftware, die im Microsoft OS/2 Presentation Manager als geräteunabhängige Schnittstelle einen signifikanten Fortschritt für die Entwicklung von Anwendungen sehen, gaben in diesen Tagen forcierte Entwicklungspläne zu dessen Unterstützung bekannt. Zu diesen Herstellern zählen Firmen wie Chips & Technologies, Texas Instruments, Western Digital Imaging, Micro Display Systems, Intel, Video-7, Moniterm und Graphic Software Systems.

Sikander Naqvi, General Manager für Grafik-Produkte bei Chips & Technologies, kommentierte die Ankündigung mit den Worten, daß »Display-Hersteller durch den MS OS/2 Presentation Manager eine wesentlich größere Flexibilität bei der Verbesserung ihrer Bildschirm-Technik haben werden. Außerdem sind Anwendungs-Entwickler nun nicht mehr gezwungen, Bildschirm-Treiber für alle auf dem Markt befindlichen Bildschirme zu entwickeln«. Statt dessen müssen die Entwickler nur noch eine Schnittstellen-Software für den Presentation-Manager schreiben, dessen Treiber den jeweiligen Bildschirm ansteuert. Die daraus entstehende Vereinfachung ist ein erheblicher Fortschritt für die Industrie. Chips & Technologies hat die Absicht, Presentation Manager-Bildschirmtreiber für alle seine Produkte anzubieten. Das Unternehmen arbeitet bereits seit zwei Jahren mit Microsoft zusammen, um seine Grafik-

Mitteilungen .... Mitteilungen .... Mitteilungen

Produkte speziell für Microsoft Windows und den MS Presentation Manager zu verbessern.

Texas Instruments und Microsoft entwickeln einen Presentation Manager-Treiber auf der Basis des weitverbreiteten Grafik-Systemprozessors 34010 und des Texas Instruments-Grafik-Architektur-(TIGA)-Interface-Standards. Nach Darstellung von K. Bala, Vizepräsident der Microprozessor/Microcontroller-Division von Texas Instruments, wird der 34010-Grafikprozessor bevorzugt für Hochleistungs-PC-Software wie Microsoft Windows und zahlreiche CAD-Anwendungen auf dem PC eingesetzt. Texas Instruments geht ferner davon aus, daß der 34010-Grafikprozessor zusammen mit dem TIGA-Interface ein Standard bei der Anwendung des Presentation Managers wird. Da die Unterstützung des Presentation Managers ein integraler Bestandteil der PC-Grafik-Strategie von Texas Instruments ist, hat sich das Unternehmen darauf festgelegt, Presentation Manager-Treiber für PCs auf 34010-Basis zu liefern. Dies wird bereits im 1. Quartal 1989 erfolgen.

Die Grafikkarten »Paradise VGA Plus«, »Paradise VGA Plus 16« und »Paradise VGA Professional« von Western Digital Imagine unterstützen ebenfalls den Presentation Manager. »Paradise Systeme stehen heute als Gütesiegel für Videoprodukte in PCs und Workstations. In diesem Sinne traten wir seit jeher für Bildschirm-Standards im Bereich der Büroautomation ein«, betont Chester A. Brown, Vice President und General Manager des Unternehmens. »Weil wir der Ansicht sind, daß der MS OS/2 Presentation Manager der nächste Standard im Bereich der Betriebssystem-Umgebungen für den Markt der Büroautomation ist, wird Western Digital Imagine weiterhin eng mit Microsoft zusammenarbeiten. Damit wollen wir sicherstellen, daß auch unsere zukünftigen IBM-kompatiblen Produkte diese Standards unterstützen«.

Micro Display Systems hat bereits mit der Auslieferung eines Presentation Manager-Treibers für ihre Ganzseiten-Bildschirme »Genius Plus« begonnen. Anwender der Genius Plus-Bildschirme können die Treiber über das »Micro Displays Electronic Bulletin Board« laden. Micro Display bietet allen MS OS/2 Systementwicklern die Genius Plus- und Genius-II-Monitore zum halben Preis an.

Charles Fox, Graphic Product Line Marketing Manager bei Intel, ist der Meinung, daß der Presentation Manager einen völlig neuen Standard in bezug auf die einfache Bedienbarkeit von PCs setzt. In Erwartung des Presentation Managers hat Intel den 82706-VGA-Controller entwickelt, um der Industrie eine 100prozentig IBM-VGA-Gate-Level-kompatible Plattform anzubieten.

Auch Paul Jain, Präsident und Geschäftsführer der Firma Video-7, geht davon aus, daß der Presentation Manager das bestimmende Grafik-Interface in den 90er Jahren sein wird. Video-7 hat deshalb die Entwicklung stark forciert, um im 1. Quartal 1989 optimierte Presentation Manager-Treiber für alle Video-7-Grafikadapter ausliefern zu können. Die VGA- und 8514/1-kompatiblen Produkte werden eine Leistungsfähigkeit und Auflösung bieten, wie sie für den effizienten Einsatz des Presentation Managers erforderlich sind.

Moniterm, ein Hersteller von Bildschirmen und Bildschirm-Controllern, wird ebenfalls den Presentation Manager umfassend unterstützen. Der mit höchster Priorität entwickelte MS OS/2 Presentation Manager-Treiber des Unternehmens soll schon Anfang '89 lieferbar sein.

Seine besten Ressourcen zur Entwicklung von Firmware und Programmierwerkzeugen für den Presentation Manager hat auch Graphics Software Systems (GSS) eingesetzt. Das Unternehmen, das Grafik-Softwarepakete und Firmware für PC-Hersteller anbietet, wird eine »Low Overhead«-Programmierumgebung für den Presentation Manager vertreiben. Daneben wird Firmware zur Verfügung gestellt, die es erlaubt, Presentation Manager-Anwendungen auf verschiedener Hardware für Hochleistungs-Grafik laufen zu lassen.

Um Peripherie-Hersteller bei der Entwicklung von Treibern für den MS OS/2 Presentation Manager zu unterstützen, bietet Microsoft ein MS OS/2 Device Driver Development Kit an, das aus Beispiel-Quellprogrammen, Debugging-Werkzeugen und einer zugehörigen Dokumentation besteht.

FormMaster: Formulargestaltung für den Profi

Die Gesellschaft für Computer-Anwendungen GCA hat mit FormMaster ein Softwarepaket für die professionelle Formulargestaltung vorgestellt. Das unter Windows implementierte Paket wurde vom Institute for Information Industry, Taipei, einem Spezialisten auf dem Gebiet der PC-Anwendersoftware, entwickelt. Die deutsche Version des Produkts für die Standard-Oberfläche MS-Windows wurde von GCA, Ludwigsburg, erstellt und ist seit November lieferbar.

FormMaster erlaubt die ökonomische Generierung von individuellen Formularen am eigenen Arbeitsplatz-PC. Das neue System sorgt bei der in vielen Unternehmen sehr aufwendigen Formulargestaltung für massive Produktivitätsfortschritte.

Ein Formular wird aus grafischen Elementen, Texten und Datenfeldern aufgebaut. Es kann ausgedruckt werden und als normales Formular genutzt werden. Die Datenfelder können jedoch auch auf dem Bildschirm ausgefüllt werden, wobei die Daten entweder über die Tastatur oder aus Dateien von Standardprogrammen eingelesen werden.

Eine übersichtliche Menü-Steuerung erlaubt es auch dem Neuling in der Windows-Welt, schnell produktiv mit dem neuen Tool zu arbeiten. Dabei wird der Anwender unterstützt durch Dateiimport-Routinen, vielfältige Editierfunktionen und eine große Zahl grafischer Darstellungsmöglichkeiten, die von horizontalen, vertikalen und diagonalen Linien beliebiger Stärke über Rechtecke bis hin

Mitteilungen .... Mitteilungen .... Mitteilungen

zu Kreisen und Ellipsen alle denkbaren Erfordernisse abdecken. Besondere Unterstützung gibt FormMaster bei Aufgaben, bei denen herkömmliche Desktop Publishing- oder Textverarbeitsprogramme versagen. Dazu gehören beispielsweise mehrfache Unterteilung von Rechtecken, Tabellen, Rechenfunktionen von Datenfeldern, etc.

WYSIWYG-Darstellung

Da es beim Entwurf eines Formblattes wichtig ist, sowohl einzelne Teile als auch das Gesamtbild direkt am Bildschirm einer Kontrolle auf das bestmögliche Design zu unterziehen, bietet FormMaster eine Multiple View-Option, mit der der Bildschirminhalt vergrößert oder verkleinert werden kann. Mit dem WYSIWYG-Interface wird eine naturgetreue Wiedergabe des Druckbildes auf dem Bildschirm ermöglicht, so daß auf kosten- und zeitintensive Probeausdrucke verzichtet werden kann.

Jedes der verwendeten Form-Elemente ist einzeln bearbeit- und abspeicherbar. Grafiken, die aus Grafik-Programmen wie Eyestar, Windows Paint oder PC Paintbrush zugeladen werden, sowie Datenfelder und Texte können einzeln oder stückweise verschoben, in der Größe verändert oder inhaltlich modifiziert werden. Die so bearbeiteten Module können auch als Komponenten in anderen Formularen verwandt werden.

Weitere Features sind die Hilfefunktion, der wahlweise Ausdruck von leeren Formularen und die Übertragung von Datei-Inhalten in fertige Formblätter.

Das anwenderfreundliche Produkt ist durch seine spezielle Ausrichtung auf ein in der betrieblichen Praxis bedeutendes Anwendungsgebiet eine Bereicherung des Spektrums professioneller Systeme für Aufgaben des Desktop Publishing.

EISA-Entwicklung verläuft nach Plan
32-Bit-EISA-Anschluß fertiggestellt

Die Gruppe der Computer-Hersteller, die am 13. September 1988 die Erweiterte Industriestandard-Architektur (EISA) ins Leben rief, kündigte an, daß alle Schlüsselelemente der EISA-Spezifikation - elektrische, mechanische und Details der Systemkonfiguration - eingearbeitet und an die am Projekt beteiligten Herstellerfirmen verteilt worden seien.

Die Ausarbeitung der EISA-Spezifikation (sie umfaßt zur Zeit über 250 Seiten) verläuft planmäßig. Marktführende Computer-Hersteller haben bereits begonnen, auf Basis dieser rechtlich nicht geschützten Industriestandard-Spezifikation Computersysteme, EISA-Chips, Zusatz-Platinen und Software-Erweiterungen zu entwickeln. Bis zum heutigen Datum haben weltweit mehr als 100 Hersteller die vollständige EISA-Spezifikation erhalten. Diese Hersteller haben das Ziel, auf EISA basierende Produkte zu entwickeln.

Erste Testversionen der wichtigsten EISA-Chips von der Intel Corp., auch »EISA-Chipset« genannt, werden voraussichtlich im zweiten Quartal dieses Jahres fertiggestellt. Erste Lieferungen der neuen Chips von Intel an die Computer-Hersteller sind für das zweite Halbjahr dieses Jahres vorgesehen. Komplette auf EISA basierende PC-Systeme und Zusatzprodukte werden Ende des Jahres verfügbar sein.

EISA-Anschluß fertiggestellt

Die Spezifikation enthält nun z.B. auch die Fertigstellung der mechanischen Details für den EISA 32-Bit-Anschluß. Dieser neue Anschluß ermöglicht die Installation von 32-Bit Hochleistungs-Erweiterungsplatinen in den neuen EISA-PCs, die gegen Ende des Jahres verfügbar sein werden.

Da EISA eine Übermenge des aktuellen 8-/16-Bit-Industriestandard-Erweiterungsbus (ISA) ist, schützt EISA sowohl im professionellen als auch im privaten Anwendungsbereich die für mehr als 100 Milliarden Dollar getätigten Investitionen in Hardware, Software, Peripherie und Schulung. Damit ist sichergestellt, daß die Anwender auch in Zukunft auf Ihre Investitionen bauen können. Der neue Anschluß ist für zwei Betriebsmodi ausgelegt: Die Anwender können sowohl die Vielzahl von verfügbaren 8- und 16-Bit-Erweiterungssteckkarten weiter verwenden als auch neue 32-Bit-Steckkarten mit ihrer eigenen Geschwindigkeit einbauen.

So wie heute viele Anwender 8-Bit-Karten in 16-Bit-Steckplätzen betreiben, werden die Kunden noch über Jahre hinaus bestehende und neue 8- und 16-Bit-Karten kaufen und einsetzen. Tatsache ist, daß bei zahlreichen Zusatzprodukten wie z.B. Modems eine Neuentwicklung als 32-Bit-Platine keinerlei Vorteile bringen würde und daß diese daher weiterhin als 8- oder 16-Bit-Produkte angeboten werden.

Der EISA 32-Bit-Anschluß hat die gleiche Größe wie der 16-Bit-Anschluß im Industrie-Standard (ISA) und nimmt auf der Systemplatine genausoviel Platz ein. Der neue Anschluß besitzt unterhalb der 16-Bit-Kontakte eine zweite Reihe von Kontakten für 32-Bit-Daten und -Adressierung. Alle 32-Bit-Anschlüsse sind mit mechanischen Einrast-Sperren ausgestattet, die verhindern, daß die 8-/16-Bit-ISA-Platinen bis zu der unteren Kontaktreihe eingeschoben werden. Die neuen EISA 32-Bit-Erweiterungsplatinen sind mit entsprechenden Kerben ausgestattet, die die Platine an den Sperren vorbeigleiten lassen, um den Anschluß an die untere Kontaktreihe herzustellen.

Der EISA-Anschluß stellt die notwendige Stromversorgung und Masseanschlüsse sowohl für 8-/16-Bit-Platinen als auch für 32-Bit-Platinen bereit. Die Stromversorgung und Masseleitungen werden sorgfältig durch den Anschluß geführt und entsprechen den EMI-Bestimmungen bezüglich der elektromagnetischen Abstrahlungs-Interferenz.

Der Anschluß ist so konstruiert, daß Erweiterungs-Platinen problemlos auf der Systemplatine installiert werden

Mitteilungen .... Mitteilungen .... Mitteilungen

können. Bei der Installation ist etwa die gleiche Kraft aufzuwenden wie bei der Installation der zur Zeit gebräuchlichen 16-Bit ISA-Platinen.

Die Konstruktion des neuen Anschlusses ist ein Beispiel für die gemeinsamen Bemühungen von EISA-Herstellern und EISA-Anbietern. Burndy Corp., AMP Inc. und andere wichtige Verbindungs-Elemente-Hersteller haben sich bereits verpflichtet, diese Bauteile zu liefern. Größere Stückzahlen werden für das 2. Quartal erwartet. Nach Ausarbeitung der mechanischen Details des EISA-Anschlusses arbeiten die Platinen-Hersteller nun intensiv an der Entwicklung von 32-Bit EISA-Zusatzplatinen.

EISA-Merkmale

Viele Anbieter von Erweiterungs-Platinen haben die Vorteile von EISA erkannt: EISA unterstützt höhere Datenübertragungsraten, bietet eine größere Platinenfläche und zusätzliche Leistung pro Steckplatz auf der Basis einer offenen Industriestandard-Spezifikation. Die zukünftigen Produkte für EISA-Platinen werden die Vorteile der Industriestandard-Architektur, z.B. verbesserte Leistung, Anschlußmöglichkeiten und größeren Funktionsumfang, weiterhin nutzen.

EISA-Background

Zu den Gründern der EISA-Gruppe gehören die Unternehmen AST Research, Inc., Compaq Computer Corporation, Epson America, Inc., Hewlett-Packard Co., NEC Information Systems, Inc., Ing. C. Olivetti & Co., Tandy Corporation, Wyse Technology und Zenith Data Systems.

EISA ist die 32-Bit-Weiterentwicklung des 8-/16-Bit-Erweiterungsbus im Industrie-Standard. Aufgrund der größeren Bandbreite bei der Datenübertragung ermöglicht EISA den PC-Herstellern die Entwicklung von Produkten, die die zukünftigen anspruchsvollen Anwendungen wie z.B. das Arbeiten im Rechnerverbund unterstützen. Die Kompatibilität zu den derzeit 20 Millionen installierten PCs im Industrie-Standard sowie die Anschlußmöglichkeiten an Minis und Mainframes bleiben dabei erhalten.

EISA ist ein offener 32-Bit I/O-Bus, der beim Einsatz in einem entsprechenden 32-Bit-PC im Intel 80386-Prozessor entstehende Rechnerverbund-Anwendungen unterstützt, z.B. LAN-Netzwerke, Kommunikations-Gateways, Datenbank-Anwendungen mit Multi-User-Betrieb sowie Transaktionsverarbeitung. EISA bietet Speicheradressierung und Datenbus-Anschlüsse mit vollem 32-Bit-Funktionsumfang und unterstützt dadurch den Speicherbereich jenseits von 16 Mbyte. EISA bietet außerdem sowohl direkten 32-Bit-DMA als auch 32-Bit Bus-Master-Support.

Zu den zusätzlichen Funktionen gehören: programmierbare Platinen-Einstellung für die automatische Konfiguration von EISA-Platinen und die softwaregestützte Konfiguration von vorhandenen ISA- und zukünftigen EISA-Platinen, die über Schalter programmierbar sind.

Neue Produkte von 3Com auf der CeBIT'89

3Com stellt auf der diesjährigen CeBIT eine Reihe von neuen Produkten vor:

1. 3Server/402 - ein mit 320 Mbyte Plattenkapazität ausgestatteter Server der 3S/400-Familie mit Backup-Möglichkeit und externer Speichererweiterung auf rund 2 Gigabyte.
2. 3+Open Internet - eine Erweiterung der 3+Open-Produktfamilie für die Internetzwerk-Kommunikation im OS/2. 3+Open Internet erfüllt die gleichen Funktionen im OS/2 wie 3+Route im DOS. Netzwerke können transparent über Standard-Telefonleitungen, speziell zugeteilte Hochgeschwindigkeitsleitungen oder Null-Modem-Verbindungen miteinander kommunizieren. Mit hoher Geschwindigkeit sind Ethernet- und Token Ring-Pakete zwischen den Netzwerken übertragbar.
3. 3+Open Mail - eine Erweiterung der 3+Open-Familie. Sie schafft die Möglichkeit der Kommunikation (E-Mail) zwischen Apple, DOS- und OS/2-Workstations. Es ist ein Store- und Foreward-System, bei dem elektronische Post auch bei ausgeschalteter Arbeitsstation zugestellt werden kann. Über 3+Open Internet kann eine WAN-Mail-Kommunikation realisiert werden.
4. 3+Open LAN Secure - ein Programm zur Erhöhung der Netzwerk-Sicherheit mit Zugriffskontrolle über ein neues Paßwort-System.
5. EtherLink/SE - eine Netzwerk-Karte, die dem Macintosh/SE den Zugang zum Ethernet eröffnet. Sie ergänzt die vorhandenen 3Com-Macintosh-Produkte.
6. Maxess SNA-Gateway - ein Hochleistungs-Gateway für die SNA-3270-Terminal-Emulation zur Programm-zu-Programm-Kommunikation LU 6.2 (Peer-to-Peer). APIs, die als Programmschnittstellen dienen, werden unterstützt, 32 Sessionen sind gleichzeitig möglich.
7. 3+Open LAN Vision - die neue Bezeichnung für LAN View - ein LAN-Management-Werkzeug für statistische Aufgaben in der Netzwerk-Überwachung für Workstations, Server und Zugriffsrechte; basiert auf dem OS/2-Presentation-Manager.

Vorankündigung:

Im Sommer 1989 werden zwei neue Produkte aus der Kooperation von 3Com mit der französischen Firma Reseaux lieferbar sein. Sie verbessern weltweite E-Mail-Verbindungen. Die Produkte 3+Open Reach/X.400 und 3+Open Internet/X.25 können PCs, PS/2-Systeme Macintosh und andere Rechner international verbinden. Mit den Netzwerk-Betriebssystemen 3+ und 3+Open LAN Manager wird die Kommunikation mit den unterschiedlichsten E-Mail-Systemen, wie PROFS von IBM, All-In-1 von DEC, Telemail und Telenet und Atlas 400 von Transpac möglich.

Mehr Speicher mit XMS und HMA:

64 Kbyte mehr Speicher adressieren unter DOS

Jeder weiß, daß der gesamte von einem Intel 8086- oder 8088-Mikroprozessor ansprechbare Speicherbereich 1 Mbyte groß ist. Es wird auch oft angenommen, daß ein Intel 80286 oder 80386 im Real Adress Modus ebenso nur 1 Mbyte ansprechen kann, aber dies ist nicht richtig. Eine genaue Betrachtung des Aufbaus der 80286- und 80386-Mikroprozessoren zeigt, daß es möglich ist fast 64 Kbyte mehr an Speicher zu adressieren.

Mit geringem Zusatzaufwand ist es den unter DOS laufenden Programmen möglich, diesen zusätzlichen Speicher zu nutzen. Dieser Artikel zeigt die Konventionen, die zum Ansprechen dieses Extraspeichers eingehalten werden müssen, und beschreibt die auftretenden Probleme.

Der obere Speicherbereich

Dieser zusätzliche Speicher wird der obere Speicherbereich genannt (HMA oder High Memory Area). Seine maximale Größe ist 64 Kbyte minus 16 Byte. Um ihn anzusprechen, muß bei einem 80286 oder 80386 die Adreßleitung A20 im Real Adress Modus aktiviert werden. Die Prozessoren 8088 und 8086 haben beide 20 Adreßleitungen (A0 - A19), was einen ansprechbaren Speicherbereich von 2^{20} Bytes oder 1 Mbyte ergibt.

Für diese Diskussion ist die Unterscheidung zwischen *Speicherzellen* und *Speicheradressen* sehr wichtig. Speicherzellen starten mit 0 und sind der Reihe nach bis zur oberen Grenze des Speichers durchnumeriert. Es besteht eine »Eins zu Eins«-Übereinstimmung zwischen Speicherzellen und den tatsächlich vorhandenen Speicherbausteinen.

Die Speicheradressen sind dagegen ein anderer Weg, um dieselbe Speicherzelle anzusprechen. Sie bestehen aus zwei Teilen, einem 16-Bit-Segment und einem 16-Bit-Offset. Im Real Adress Modus beginnt alle 10h Bytes ein neues Segment im Speicher. Zum Beispiel beginnt das Segment 1234h an der Stelle 1234h * 10h oder 12340h des Speichers. Ist ein Offset mit angegeben, so wird er einfach zur Startadresse des Segmentes hinzuaddiert. Die Adresse 1234:5678h zeigt also auf die Speicherzelle 12340h + 5678h oder 179B8h (*Bild 1*).

Der 80286/80386 besitzt mehrere Adreßleitungen und kann, wenn er im Protected Modus arbeitet, mehr Speicher adressieren. Im Real Adress Modus ist die einundzwanzigste Adreßleitung A20 deaktiviert, womit der Speicherbereich auf genau die Größe wie beim 8088 oder 8086 beschränkt ist. Jedesmal wenn der Prozessor auf Speicherbereiche über 1 Mbyte zugreifen will, läuft er auf den Beginn des Speichers (Adresse 0) über.

Wenn sich ein Computer wie der IBM PC/AT oder der Compaq 386 im Real Adress Modus befindet, können Sie die Adreßleitung A20 aktivieren oder deaktivieren.

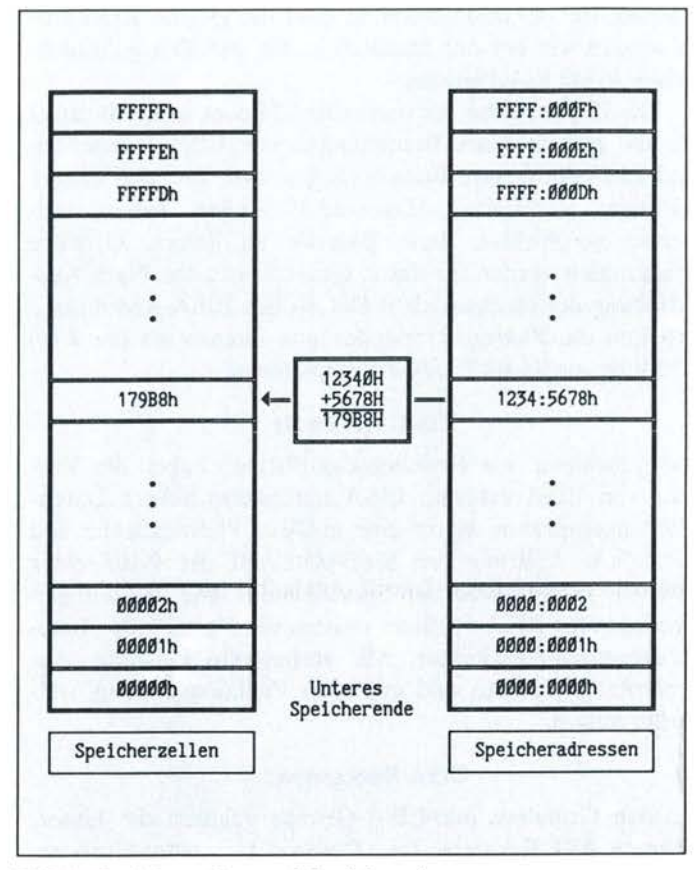


Bild 1: *Speicherzellen und Speicheradressen.*

Ist die Adreßleitung A20 aktiviert, führen Versuche, den Speicher über 1 Mbyte anzusprechen, nicht zu einem Überlauf auf den Anfang. Vielmehr ist es möglich, über die 1-Mbyte-Grenze in den Extended Speicher zu gelangen.

Verdeutlichen Sie sich das folgende Beispiel. Bei der Benützung der Intel Real Modus-Adressierung zeigt die Adresse FFFF:0Fh auf die Speicherzelle FFFFFh, die gerade 1 Byte unter der Mbyte Grenze liegt (*Tabelle 1A*). Dies ist die Obergrenze, die mit einer 20-Bit-Adressierung erreicht werden kann. Die Adresse FFFF:10h zeigt logischerweise auf die Adresse 100000h, welches das Byte an der Mbyte Grenze ist. Da wir aber nur 20 Bits gültige Informationen haben, wird die 1, die an der 21. Position (A20) ist, ignoriert. Daher zeigt die Adresse auf die Speicherzelle 0h (*Tabelle 1B* und *Bild 2*).

Wird die Speicherleitung A20 aktiviert, gibt es keinen Überlauf, da das benötigte 21. Adreßbit zur Verfügung steht. Dies bedeutet, daß die Adresse FFFF:10h tatsächlich der Adresse 100000h entspricht (*Tabelle 1C*).

Die Konventionen für die Adressierung im Real Adress Modus enden mit der Adresse FFFF:FFFFh, was 10FFEFh entspricht. So enden wir 64 Kbyte minus 16 Byte über der Mbyte-Grenze (*Tabelle 1D*). Dieser zusätzliche Speicher wird High Memory Area (oberer Speicherbereich) genannt (HMA).

A)
```
FFFF:000FH =
        (FFFFH*10H)+000FH   =
        FFFF0H+0000FH       =
        FFFFFH              =
         11111111111111111111B [mit 20 Bits]
```
B)
```
FFFF:0010H =
        (FFFFH*10H)+0010H   =
        FFFF0H+00010H       =
        100000H             =
         00000000000000000000B [mit 20 Bits]
```
(Bewirkt Überlauf zurück an Speicherzelle 0000:0000)

C)
```
FFFF:0010H =
        100000000000000000000B [mit 21 Bits]
```
(Bewirkt erweiterteten Zugriff oberhalb 1 Mbyte)

D)
```
FFFF:FFFFH =
        (FFFFH*10H)+FFFFH   =
        FFFF0H+0FFFFH       =
        10FFFEFH            =
```

Tabelle 1A bis 1D: *Adressierungskonventionen von Intel.*

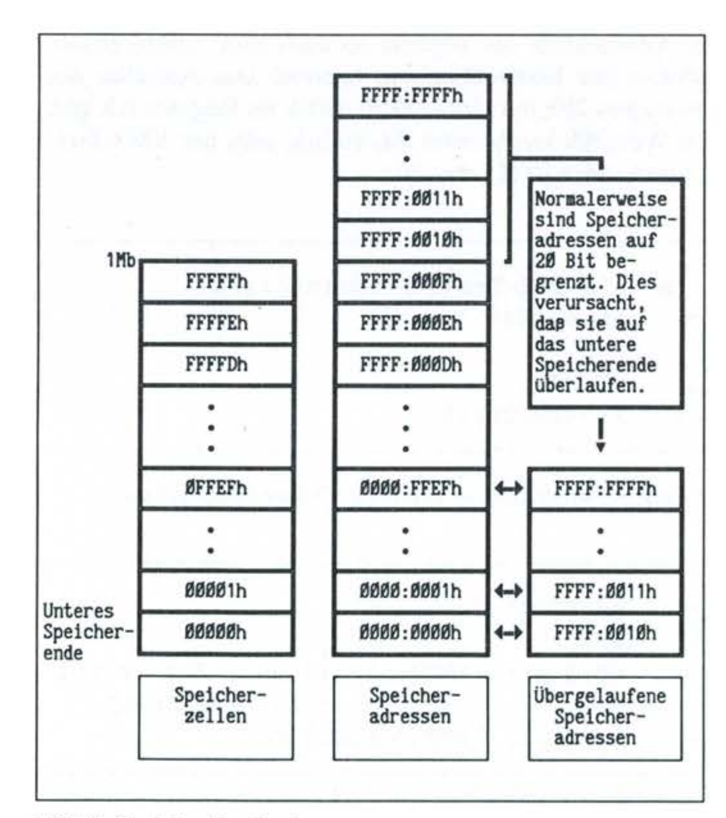


Bild 2: *Speicherüberlauf*

HMA-Besonderheiten

Einige Aspekte der HMA unterscheidet diesen Bereich von normalen Speichersegmenten. Während normale Speichersegmente immer verfügbar sind, kann der obere Speicherbereich nur angesprochen werden, wenn die Adreßleitung A20 aktiviert ist. Ein weiterer Unterschied ist, daß dieser Bereich nicht an einer Segmentgrenze beginnt, sondern 16 Byte nach dem Segment FFFFh.

Der wichtigste Unterschied ist aber, daß dieser Bereich nicht unterteilt werden kann, da kein Segment mehr in diesem Bereich beginnt. Deshalb kann dieser Bereich nicht von mehreren verschiedenen Programmen zur selben Zeit genutzt werden.

In der Theorie ist die Nutzung des HMA ganz einfach. Sie brauchen nur ein Programm in den ersten Teil des Expanded Memory verschieben, die Adreßleitung A20 aktivieren und zu einer Speicherzelle über der Adresse FFFF:10h springen. Einige Besonderheiten lassen es aber ratsam erscheinen, Einschränkungen zu machen.

HMA-Probleme

Im folgenden werden nähere Betrachtungen der HMA-Besonderheiten angestellt.

Ansprechen der A20-Adreßleitung: Die Vorgehensweise des Aktivierens der Adreßleitung A20 weicht von Computer zu Computer ab. Einige Computer verfügen nicht einmal über diese Adreßleitung. Zu unterscheiden, welchen Computer Sie verwenden, und wie es möglich ist, die Adreßleitung A20 anzusprechen, ist ein schwieriges Unternehmen. Deshalb ist eine einheitliche, vom Computer unabhängige Möglichkeit gefordert, um diese Ressource anzusprechen.

Belegen des HMA: Den oberen Speicherbereich muß man sich als völlig ungeschützte Ressource vorstellen. Da er nur in einem Segment sein kann, kann er nicht auf mehrere Programme zur selben Zeit verteilt werden. Da DOS kein Multitasking-Betriebssystem ist, kann es sowohl TSR-Programme als auch Einheitenreiber geben, die Konflikte verursachen können, wenn sie diese Ressource belegen. Deshalb ist ein Schema für das Belegen und Freigeben der HMA erforderlich.

Schutz der HMA vor versehentlichem Überschreiben: Einige TSR-Programme und Einheiten-Treiber, wie zum Beispiel IBMs VDISK-Programm, belegen einfach den unteren Teil der HMA und legen dort ihre Daten ab. Ist einer dieser Treiber installiert, kann die HMA nicht genutzt werden. Andere Treiber, wie etwa ältere Versionen von Microsofts RAMDrive, allokieren Extended Memory von oben nach unten. Solange sie 64 Kbyte freilassen, ist die HMA verfügbar. Ist aber die HMA aktiv, so muß sie vom Überschreiben durch solche Programme geschützt werden.

Feststellen der Existenz der HMA: Es müssen mindestens 64 Kbyte Extended Memory zur Verfügung stehen, um die HMA zu unterstützen. Dieser Zustand muß zuerst über-

prüft werden. Ein weiterer Test muß unternommen werden, um festzustellen, ob die Adreßleitung A20 aktiviert und deaktiviert werden kann.

Priorisierung der HMA Benutzung: Wenn sich ein DOS-Einheiten-Treiber oder ein TSR-Programm in die HMA verschiebt, wird der Speicherbereich, den dieses Programm unter der 640-Kbyte-Grenze belegen würde, für andere Programme frei. Stellen Sie sich den Zustand vor, wenn zwei TSR-Programme, die die HMA benutzen könnten, installiert werden. Bedenken Sie, daß nur ein Programm jeweils die HMA benutzen kann. Nehmen wir an, das erste Programm kann 50 Kbyte in die HMA verlegen, das zweite hingegen 62 (*Bild 3*). Wenn das zweite in die HMA verschoben wird, so sind zusätzlich 12 Kbyte im Speicher unter 640 Kbyte frei. Der Benutzer benötigt eine Möglichkeit, um aus einer solchen Situation das Beste zu machen.

Die Besonderheiten der HMA bereiten dem DOS-Programmierer einige lösbare Probleme. Als Microsoft diesen Speicherbereich zuerst nutzen wollte, erkannte die Firma, das eine Standardmethode zum Ansprechen des Speichers entwickelt werden muß, bevor dieser Bereich größere Nutzung finden kann.

XMS als Lösung

Im Juli 1988 veröffentlichten Microsoft, Intel, Lotus und andere Softwarehäuser die DOS Extended Memory Spezifikation (XMS) Version 2.0. XMS erlaubt es den DOS Programmierern, alle Teile des Extended Memory, einschließlich der HMA, effizient zu nutzen. Ein XMS-Treiber löst zwei der wichtigsten HMA-Probleme, das maschinenunabhängige Umschalten der Adreßleitung A20 und das gleichzeitige Nutzen der HMA von mehreren Programmen.

Zusätzlich zur HMA-Unterstützung erlaubt XMS Programmen, große Mengen von Daten im Extended Memory zu speichern. Der Rest dieses Artikels bezieht sich aber nur auf den HMA-spezifischen Teil von XMS. Programmierer können die komplette Dokumentation und den HIMEM.SYS-Treiber von Microsoft beziehen.

Nutzung der HMA

DOS-Programmierer müssen vor der Nutzung der HMA folgendes unternehmen:

- Feststellen, ob ein XMS-Treiber vorhanden ist.
- Adresse der Kontrollfunktion des Treibers feststellen.
- HMA anfordern.

Erlaubt der XMS-Treiber Zugriff auf die HMA, so kann das Programm:

- die Adreßleitung A20 aktivieren,
- bis zu 64 Kbyte Programm in die HMA kopieren,
- ausführen,
- die Adreßleitung A20 deaktivieren,
- HMA freigeben und
- beenden.

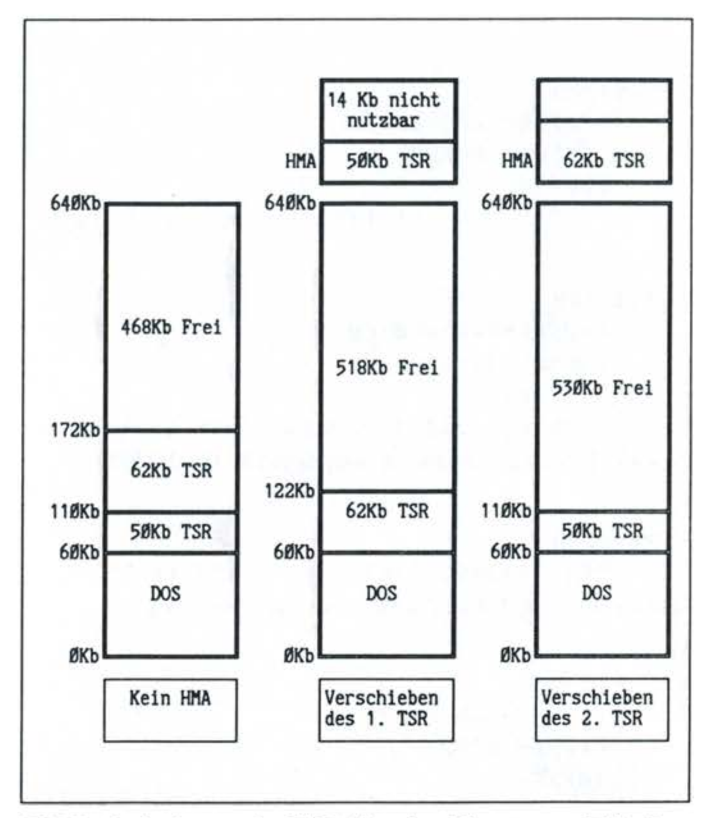


Bild 3: *Optimierung der HMA für den Einsatz von TSR-Programmen.*

Der Multiplexinterrupt 2Fh wird sowohl zum Feststellen der Anwesenheit des Treibers als auch zum Anfordern der Adresse der Kontrollfunktion benützt. Das Aufrufen des Interruptes 2Fh mit einem Wert 4300h im Register AX gibt den Wert 80h im Register AL zurück, falls der XMS-Treiber vorhanden ist (*Listing 1*).

```
; Ist ein XMS-Treiber installiert?
mov    ax,4300h
int    2Fh
cmp    al,80h
jne    KeinXMSTreiber
```

Listing 1: *Feststellen, ob ein XMS-Treiber installiert ist.*

```
mov    ax,4310h
int    2Fh
mov    word ptr [XMSControl],bx ; XMSControl
                                ; ist ein DWORD
mov    word ptr [XMSControl],es
```

Listing 2: *Die Kontrollfunktion des XMS-Treibers herausfinden.*

```
; Hole die Versionsnummer des XMS-Treibers
mov     ah,00h
call    [XMSControl] ; Hole XMS-Versionsnummer
```

Listing 3: *Aufruf einer XMS-Funktion.*

Funktion	Beschreibung
00h	Hole XMS-Versionsnummer
01h	HMS anfordern
02h	HMS freigeben
03h	A20 global ermöglichen
04h	A20 global sperren
05h	A20 lokal ermöglichen
06h	A20 lokal sperren
07h	A20 abfragen
08h	Freien Extended Memory abfragen
09h	Extended Memory Block allokieren
0Ah	Extended Memory Block freigeben
0Bh	Extended Memory Block verschieben
0Ch	Extended Memory Block sperren
0Dh	Extended Memory Block lösen
0Eh	Hole Handle-Information
0Fh	Extended Memory Block reallokieren
10h	Oberen Speicherblock anfordern
11h	Oberen Speicherblock freigeben

Tabelle 1: *Die Funktionen der XMS-Programmierschnittstelle.*

Das Aufrufen des Interrupts 2Fh mit einem Wert 4310h im Register AX gibt die Adresse der Kontrollfunktion des Treibers in der Registerkombination ES:BX zurück (*Listing 2*). Programme können die XMS-Funktionen aufrufen, indem sie das Register AH mit einem der 18 XMS-Funktionsnummern laden und dann die Kontrollfunktion aufrufen (*Listing 3*). Für den Zugriff auf die HMA werden nur die Funktionen 0 bis 4 benötigt. *Tabelle 1* zeigt die 18 XMS-Funktionen.

DOS Applikationsprogramme können die HMA anfordern, indem sie die Kontrollfunktion mit AH = 01h (Request HMA) und DX=FFFFh aufrufen. TSR-Programme müssen das Register DX unterschiedlich vorbelegen, wie wir später noch sehen werden. Kehrt der Funktionsaufruf mit dem Wert 1 im Register AX zurück, ist der Zugriff auf die HMA erlaubt. Sie können dann die Funktion 3 (GlobalEnableA20) aufrufen, um die Adreßleitung A20 des Computers zu aktivieren. Nachdem die Adreßleitung erfolgreich aktiviert wurde, kann das Programm den zusätzlichen Speicherbereich nach Belieben nutzen. Normalerweise wird es möglichst viele Teile seines Programms oder der Daten verschieben, um Speicherbereich unter 640 Kbyte für dynamische Programmteile oder Daten freizubekommen.

Am Programmende muß das Programm die XMS-Funktion 4 (GlobalDisableA20) aufrufen, um die Adreßleitung A20 zu deaktivieren. Einige ältere DOS-Programme vertrauen auf den Adreßüberlauf, und bringen das System zum Absturz, wenn die Adreßleitung A20 aktiviert ist, während sie laufen. Vor dem Ende muß das Programm die HMA noch freigeben, indem es die XMS-Funktion 2 (Release HMA) aufruft.

Priorisierung der HMA

Lassen Sie uns zu dem Problem der Priorisierung des HMA zurückkommen, wenn mehrere TSR installiert sind, die alle auf die HMA zurückgreifen. Der XMS-Treiber wird durch eine Zeile ähnlich

`DEVICE=HIMEM.SYS`

in der Datei CONFIG.SYS installiert. Diese Zeile darf einen optionalen Parameter

`/HMAMIN=`

enthalten. Dieser Parameter gibt die minimale Speichergröße an, die ein TSR-Programm anfordern muß, um Zugang zur HMA zu erhalten.

Ein TSR-Programm muß zusätzlich vor dem Aufruf der XMS-Funktion 1 (Request HMA) die Größe des benötigten HMA-Speichers in Bytes im Register DX angeben. Der XMS-Treiber vergleicht diese Größe mit der Angabe von /HMAMIN=. Ist die Anforderung größer, bekommt das TSR-Programm die HMA zugeteilt. Ist zum Beispiel

`DEVICE=HIMEM.SYS /HMAMIN=55`

in der Datei CONFIG.SYS enthalten, so erhalten nur Programme, die 55 oder mehr Kbyte anfordern, Zugang zur HMA. Beachten Sie, daß dieses Problem nicht die DOS-Applikationen betrifft, da sie die HMA freigeben, bevor sie beenden. Da sie sich nicht um die Priorität kümmern müssen, sollten sie das Register DX mit FFFFh laden bevor sie die Funktion 1 (Request HMA) des XMS aufrufen. Hierdurch umgehen sie die Prüfung.

Da einige Programme sich auf den Adreßüberlauf verlassen, müssen TSR-Programme, die die HMA benutzen, die Adreßleitung A20 deaktivieren, bevor sie die Kontrolle wieder abgeben. Das erfordert, daß diese Programme eine kleine Routine unter der 640-Kbyte-Grenze haben müssen, die die Adreßleitung A20 aktiviert und dann den Programmteil in der HMA anspringt, wenn das TSR-Programm aktiviert wird. Beim Beenden des TSR-Programms muß es eine andere kleine Routine unter der 640-Kbyte-Grenze anspringen, die die Adreßleitung A20 deaktiviert und dann die Kontrolle abgibt.

HIMEM.SYS

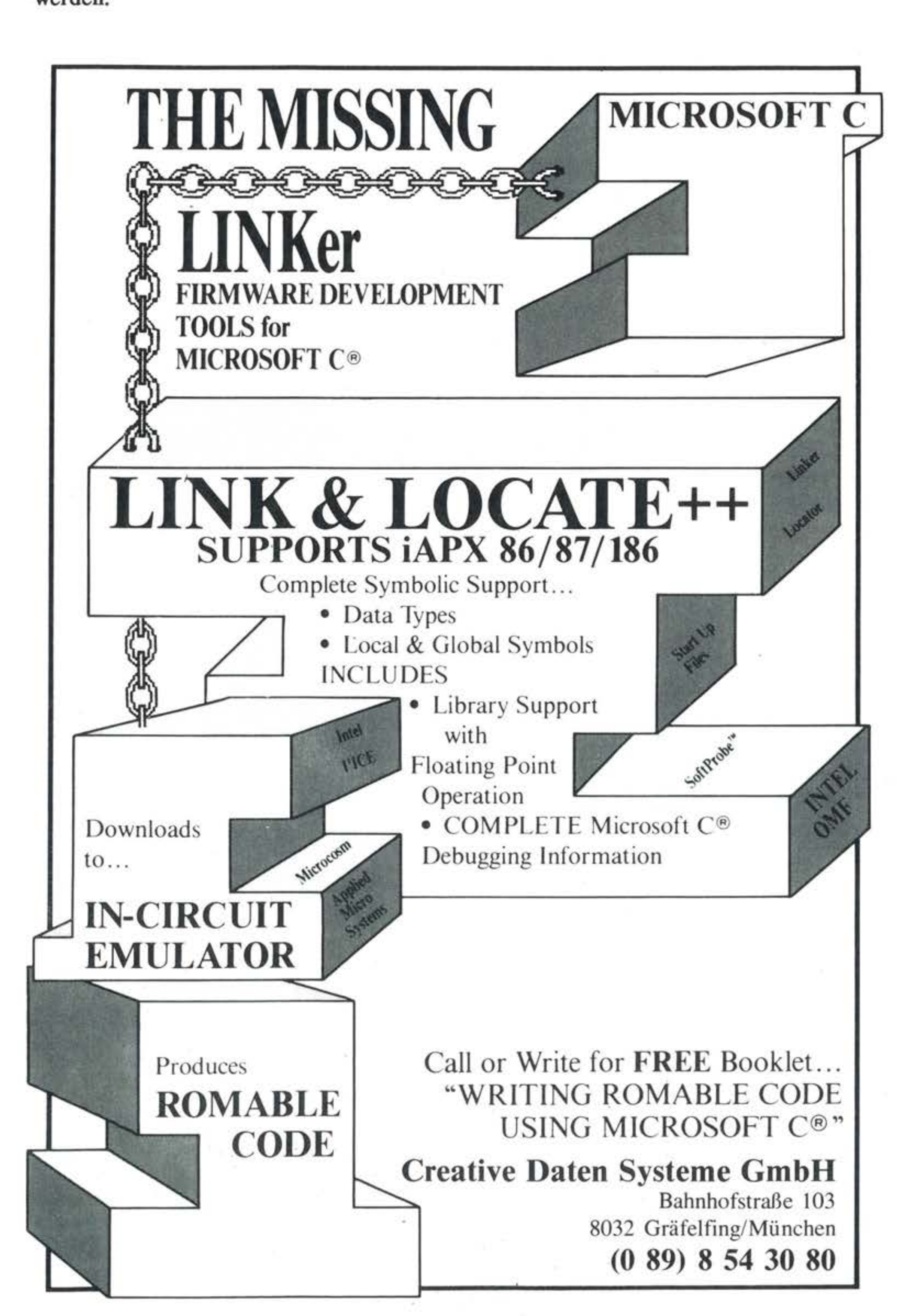
Microsofts XMS-Treiber HIMEM.SYS ist in dem XMS Developers Kit enthalten. HIMEM.SYS Version 2.04 ist ein vollständiger XMS-Treiber, der Unterstützung bei der Umschaltung der Adreßleitung A20 für eine Reihe von

Computern bietet, einschließlich der Familien IBM PC und PS/2 und der Kompatiblen.

Eine frühere Version von HIMEM.SYS (Version 1.1) ist auch im Lieferumfang von Microsoft Windows/286 und Microsoft Windows/386 der Version 2.1 enthalten. Dieser Treiber unterstützt nur die ersten 9 der 18 XMS-Funktionen, was bedeutet, daß alle HMA-Funktionen unterstützt werden.

Die Spezifikation des XMS Programmers Applications Interface erlaubt die Programmierung von zusätzlichen 64 Kbyte an Speicher auf vielen Computern, ohne damit zusätzliche Hardware zu benötigen. Richtig angewendet kann die HMA die Geschwindigkeit deutlich verbessern und die Datenmenge vergrößern, die ein Programm bearbeiten kann.

Chip Anderson



Bücher über den neuen Betriebssystemstandard und den Textverarbeitungsbestseller:

Bücher zu OS/2 und Microsoft Word

Wir haben uns diesmal mit unseren Buchbesprechungen auf die beiden Themen OS/2 und Microsoft Word konzentriert, wobei wir Bücher ausgesucht haben, die sich hauptsächlich mit den Problemen des Anwenders beschäftigen, denn auf den Gebieten OS/2 und Word müssen auch viele Programmierer die Bedienung erst noch lernen.

OS/2

Ein reines Buch für den OS/2-Anwender hat Frank Piefke mit dem »OS/2-BS/2: Anwenderbuch« vorgelegt, so konnte er sich denn auch den geschichtlichen Einstieg nicht verkneifen. Danach folgt dann in acht logisch gegliederten Kapiteln die übersichtliche Beschreibung der Möglichkeiten und Befehle von OS/2 Version 1.0. Was für den Anfänger dabei zu schwierig oder unwichtig sein könnte, wurde in Kursivschrift gedruckt. Obwohl es sich bei diesem Buch nicht um eine reine Schulung handelt, befinden sich am Ende jedes Kapitels ein paar Aufgaben, die eine Selbstprüfung des Kenntnisstands ermöglichen.

Frank Piefke: »OS/2-BS/2: Anwenderbuch«, Heidelberg, Hüthig Verlag, 1988; 424 Seiten; ISBN 3-7785-1610-8; DM 58,-.

Die beiden nun folgenden, ähnlich angelegten Bücher aus der Düsseldorfer Szene gehen da etwas weiter. Die beiden Autoren Schieb und Tischer (letzterer den MSJ-Lesern sicherlich wohlbekannt) gehen nach einer Vorstellung von OS/2 Version 1.0 und der Installation desselben auf alle Befehle im einzelnen ein, um dann auf die Stapelverarbeitung und die Konfiguration des System zu kommen. Als wichtiger Bestandteil des Buches werden aber auch die technischen Grundlagen und die Programmierung unter OS/2 beschrieben.

Jörg Schieb, Michael Tischer: »Das große Buch zu OS/2«, Düsseldorf, Data Becker, 1988; 460 Seiten; ISBN 3-89011-212-9; DM 49,-.

Einen ähnlichen Aufbau hat diese aus dem amerikanischen übersetzte Beschreibung von OS/2 *»Arbeiten mit OS/2 & BS/2«*. Sie reißt die Programmierung unter OS/2 jedoch nur an. Besonders nutzbringend wurden die Buchdeckel gestaltet: klappt man das Buch vorn oder hinten auf, findet man eine kurze Beschreibung der OS/2-Befehle mit Syntaxangabe.

J. Robbins, M.A. Beisecker, W. Schellenberger: »Arbeiten mit OS/2 & BS/2«, Düsseldorf, Sybex Verlag, 439 Seiten; ISBN 3-88745-656-4; DM 59,-.

Microsoft Word

Dies ist keine Buchbesprechung im eigentlichen Sinne, sondern nur eine Empfehlung: Wie könnte ich als Autor (oder auch mein Redaktionskollege Jürgensmeier) das Word-Lexikon objektiv beurteilen?! Dennoch wollen wir Ihnen die Existenz dieses Buches nicht vorenthalten.

Das Lexikon ist, wie der Name schon andeutet, alphabetisch aufgebaut. Es bietet auf 631 Seiten zu (nahezu) jedem Problem, das bei der Arbeit mit Word 4.0 auftreten kann, ein Stichwort mit ausführlichen Erklärungen. Dabei erleichtern die übersichtliche Gliederung und die vielen Beispiele die schnelle Lösung eines Problems. Breiten Raum nehmen vor allem die oft vernachlässigten Themen Druckformatvorlagen und Makros ein, und auch auf die Beschreibung der Fehlermeldungen wurde großer Wert gelegt. Wenn man den Aussagen eines Leserbriefs glauben schenken darf, dann wird im diesem Buch »klar und erschöpfend die ja doch nicht ganz einfache Problematik von Word 4.0 handlich und praktikabel dargestellt«.

Hartmut Niemeier, Marianne Nuß: »Word 4.0 Lexikon«, München: Markt&Technik Verlag, 1988; 631 Seiten; ISBN 3-89090-621-4; DM 79,-.

Einen engeren Bereich von Word 4.0 betrachtet das Autorenteam Förster/Zwernemann: Hauptsächlich sind es die Makroprogrammierung, Druckformatvorlagen, Textbausteine, Formularerstellung, Serienbriefe und der Druck von Etiketten.

Die einzelnen Themen werden Schritt für Schritt erklärt und können so leicht nachvollzogen werden. Auf der mitgelieferten Diskette befinden sich außerdem nahezu alle Druckformate sowie die drei großen Makroprogramme datenVERWALTER, etikettenASSISTEMT und reisekostenVERWALTER, die sofort in Gebrauch genommen werden können.

Hans-Peter Förster, Martin Zwernemann: »Makroprogramme, Standardformate und Musterformulare mit Word 4.0«, Würzburg: Vogel Verlag, 1988; 144 Seiten; inkl. Diskette; ISBN 3-8023-0246-X; DM 48,-.

Noch mehr ins Detail geht Gabi Broszat in ihrem Buch *»Word Makros für die tägliche Arbeit mit MS-Word«*: Sie beschränkt sich allein auf die Darstellung der Makros. Dabei wurde das Buch so angelegt, daß auch der blutigste Laie den großen Nutzen von Makros erkennen wird und nach kurzer Zeit eigene Makros zumindest aufzeichnet.

Doch auch die Makro-Kenner kommen nicht zu kurz. Sie werden ihre Freude sowohl an den kleinen nützlichen Makros haben als auch die komplizierten auf den hinteren Seiten als Anregungen für eigene Makros hernehmen. Im übrigen sind auf der beigelegten Diskette alle Makros einsatzfertig für den sofortigen Einsatz enthalten.

Gabi Broszat: »Word Makros für die tägliche Arbeit mit MS-Word«, München: Systhema Verlag, 1989; ca. 180 Seiten; inkl. Diskette; ISBN 3-89390-303-8.

PC Intern:

Die Speicherverwaltung von DOS

In diesem Auszug aus dem Buch »PC Intern« erläutert Michael Tischer die Speicherverwaltung und den Aufbau der Memory Control Blocks von DOS.

Aus der Sicht von DOS unterteilt sich der maximal 640 KByte große Hauptspeicher eines PC in zwei (logische) Bereiche. Der erste Bereich dient ihm selbst und wird deshalb als Betriebssystembereich bezeichnet. Er beginnt an der untersten RAM-Speicherstelle, an der Adresse 0000:0000 und enthält neben der Interrupt-Vektortabelle interne Tabellen, Puffer, Variablenspeicher und den residenten Betriebssystemcode. Dazu zählen unter anderem die fest in das System eingebundenen Geräte-Treiber sowie das Systemmodul, das den Programmcode für die DOS-Funktionen enthält, die über den Interrupt 21h aufgerufen werden können. Die Größe dieses Bereichs hängt von der DOS-Version, der Größe der installierten Gerätetreiber und anderen Faktoren wie z.B. der Anzahl der Plattenpuffer ab.

Der zweite, weitaus größere Bereich wird als Transient Program Area (TPA) bezeichnet. Er nimmt die auszuführenden Programme nebst den dazugehörigen Environment-Blöcken auf und hat seinen Anfang unmittelbar hinter dem Betriebssystembereich. Je nach den Speicheranforderungen der einzelnen Programme teilt DOS ihnen einen unterschiedlich großen Speicherbereich zu. Um die so allokierten Speicherbereiche zur verwalten, stellt DOS ihnen jeweils einen Datenblock voran, der in der DOS-Terminologie als Memory Control Block (MCB) bezeichnet wird. Er ist 16 Bytes (ein Paragraph) groß, beginnt immer an einer durch 16 teilbaren Adresse und geht dem allokierten Speicherbereich unmittelbar voraus. Zwar arbeitet DOS bei den Funktionen zur Speicherverwaltung immer mit der Segmentadresse des allokierten Bereichs, doch läßt sich die Segmentadresse des zugehörigen MCB leicht ermitteln, indem man einfach 1 von der Segmentadresse des Speicherbereichs abzieht.

Aufbau eines Memory-Control-Blocks (MCB) im Speicher		
Addr	Inhalt	Typ
+00h	ID ("Z" = letzter MCB, "M" = es folgen weitere)	1 BYTE
+01h	Segmentadresse des zugehörigen PSP	1 WORD
+03h	Anzahl der Paragraphen im allokierten Speicherbereich	1 WORD
+05h	ungenutzt	11 BYTE
+10h	der allokierte Speicherbereich	x PARAG.
Länge: 16 + die Größe des allokierten Speicherbereichs		

Abbildung 1: *Aufbau eines MCB.*

Wie die *Abbildung 1* zeigt, enthält der MCB drei Felder. Im ersten Feld hat sich einer der Schöpfer von MS-DOS, der Amerikaner Mark Zbikowski, verewigt, indem es immer einen der beiden Buchstaben seiner Initialen enthält. "M" zeigt dabei an, daß auf diesen MCB noch weitere MCBs folgen, während "Z" darauf hindeutet, daß es sich hier um den letzten MCB im Speicher handelt.

Im zweiten Feld ist die Segmentadresse des PSP des zugehörigen Programms verzeichnet. Sie ist nur dann von Bedeutung, wenn es sich bei dem allokierten Speicherbereich um das Environment eines Programms handelt, auf dessen PSP dieses Feld dann zeigt und so eine Verknüpfung herstellt. Handelt es sich bei dem Speicherbereich hingegen um einen PSP, so zeigt dieses Feld in den meisten Fällen auf den Speicherbereich selbst.

Bedeutend wichtiger ist demgegenüber das dritte Feld im MCB, das die Größe des zugehörigen Speicherbereichs in Paragraphen angibt. Da unmittelbar hinter dem Ende des allokierten Speicherbereichs der nächste MCB folgt (sofern das ID-Feld nicht "Z" enthält), enthält dieses Feld damit auch die Entfernung zum nächsten MCB minus 1. Da somit jeder MCB indirekt auch auf den folgenden MCB zeigt, ergibt sich dadurch eine verkettete Liste, mit deren Hilfe alle MCBs aufgespürt werden können.

Wird ein Programm von dem EXEC-Loader des DOS geladen und zur Ausführung gebracht, werden von dieser Funktion zunächst zwei Datenbereiche über eine andere DOS-Funktion angefordert. Der erste dieser beiden Bereiche dient dabei zur Aufnahme des Environment-Block, während der zweite das eigentliche Programm und den dazugehörigen PSP aufnehmen soll. Gerade aber die Größe dieses Bereiches, d.h. die Größe des Speicherraums, der einem Programm zur Verfügung gestellt werden muß, ist für den EXEC-Loader nur sehr schwer abzuschätzen.

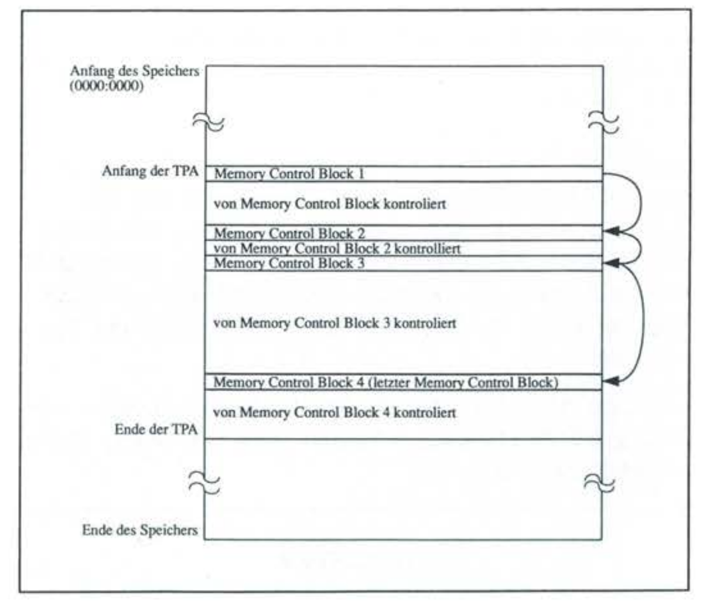


Abbildung 2: *Die Verwaltung des Speichers mit Hilfe von Memory Control Blocks*

Dies fällt ihm bei COM-Programmen noch schwerer als bei EXE-Programmen, da sie auf Platte als Abbild des Speicherinhalts gespeichert sind und ihnen keinerlei Informationen vorausgehen. Aus diesem Grund nimmt DOS den Maximalfall an und belegt für ein COM-Programm den gesamten zur Verfügung stehenden Speicher.

Diese Methode war zwar in den Anfangstagen des DOS recht effizient, bringt aber heute einige Probleme mit sich. Während sich nämlich in den ersten DOS-Versionen immer nur ein Programm im Speicher befinden konnte, ist es mittlerweile möglich und gebräuchlich, daß ein Programm ein anderes Programm in den Speicher lädt und zur Ausführung bringt, bzw. daß bestimmte Programme im Speicher resident verankert werden. Dies ist aber nicht möglich, wenn, wie z.B. nach dem Laden eines COM-Programms, kein Speicher mehr zur Verfügung steht. Aus diesem Grund sollte ein COM-Programm nach seinem Start immer den Speicher, den es nicht benötigt, wieder freigeben.

Ein COM-Programm kann allerdings nur dann geladen werden, wenn genügend Speicher frei ist, d.h. der freie Speicherplatz mindestens so groß wie das COM-Programm plus 256 Bytes für den PSP und mindestens 2 Bytes für den Stack ist. Allerdings muß das COM-Programm dafür Sorge tragen, daß der ihm zur Verfügung gestellte Speicher auch ausreicht. Gerade unter den oben erwähnten Minimalbedingungen ist die fehlerfreie Abarbeitung sehr in Frage gestellt, da wohl kaum ein Programm mit nur 2 Bytes Stack auskommt.

Ganz anders verhält es sich mit EXE-Programmen, denen auf der Platte eine ganze Reihe von Informationen vorangehen, die vom Linker ermittelt werden. Diesen Informationen kann der EXEC-Loader entnehmen, wieviel Speicherraum für die Segmente Code, Daten und Stack reserviert werden muß. Daneben enthält der Vorspann eines EXE-Programms weitere Informationen, die angeben, wieviel zusätzlicher Speicher für das EXE-Programm bereitgehalten werden muß. Dabei wird jedoch nicht die genaue Anzahl der Bytes angegeben, sondern vielmehr eine Ober- und eine Untergrenze des weiterhin benötigten Speichers definiert. Nach Möglichkeit versucht der EXEC-Loader, die Obergrenze an Speicher zu reservieren. Ist dies nicht möglich, gibt er sich auch mit der Untergrenze zufrieden bzw. reserviert den Rest des verbleibenden Speichers. Kann jedoch auch die Untergrenze an Speicher nicht zugeteilt werden, wird der Ladevorgang abgebrochen und die Kontrolle an das Programm zurückgegeben, das den EXEC-Loader aufgerufen hatte (in den meisten Fällen ist dies der Kommandoprozessor).

Gleiches geschieht nach der Ausführung des Programms, wobei jedoch zuvor der EXEC-Loader, wenn nicht durch einen bestimmten Funktionsaufruf durch das ausgeführte Programm (Funktion 31H des Interrupts 21H) verhindert, den reservierten Speicherbereich zur weiteren Verwendung wieder freigibt.

Nachdem wir uns den theoretischen Hintergrund erarbeitet haben, wollen wir uns nun die Funktionen des DOS im Speichermanagement anschauen. Es sind dies im einzelnen die Funktionen 48h, 49h und 4Ah, die über den Interrupt 21h aufgerufen werden, wobei jeweils die Funktionsnummer im AH-Register übergeben wird.

Funktion 48h dient dabei zur Reservierung von Speicherplatz. Ihr wird neben der Funktionsnummer im AH-Register die Anzahl des reservierenden Paragraphen (jeweils 16 Bytes) im BX-Register übermittelt. Konnte die angeforderte Anzahl an Paragraphen reserviert werden, kehrt die Funktion mit einem gelöschten Carry-Flag zurück. Das AX-Register gibt dann die Segmentadresse des reservierten Speichers an. Er beginnt somit an der Adresse AX:0000. (Einen Paragraph davor befindet sich der zugehörige MCB.) Konnte jedoch nicht soviel Speicher wie gewünscht zur Verfügung gestellt werden, ist das Carry-Flag nach dem Funktionsaufruf gesetzt. Das AX-Register enthält dann einen Fehlercode und das BX-Register die Größe des maximal noch verfügbaren Speichers in Paragraphen.

Da das DOS keine Funktion zur Verfügung stellt, mit deren Hilfe die Größe des noch nicht allokierten Speichers ermittelt werden kann, wird diese Funktion von Programmen oft in diesem Sinne mißbraucht. Indem vor ihrem Aufruf der Wert 0FFFFh in das BX-Register geladen wird, wird ein Speicherbereich mit der Größe von 1 MByte angefordert, den das DOS natürlich nicht allokieren kann. Dadurch aber liefert die Funktion automatisch im BX-Register die Anzahl der noch nicht allokierten Paragraphen zurück.

Das Gegenstück zu der Funktion 48h stellt die Funktion 49h dar, mit der durch die Funktion 48h reservierter Speicherbereich wieder freigegeben werden kann. Ihr wird deshalb im ES-Register die Segmentadresse des freizugebenden Speicherbereichs übergeben, die zuvor beim Aufruf der Funktion 48h im AX-Register erhalten worden war. Normalerweise sollte die Funktion 49h fehlerfrei ausgeführt werden können und dadurch das Carry-Flag nach dem Funktionsaufruf gelöscht sein. Ist dem jedoch nicht so, kann das zum einen daran liegen, daß der MCB des Speicherbereichs (durch ein Programm) zerstört worden ist, oder daß die im ES-Register übergebene Segmentadresse nicht zu einem reservierten Speicherbereich gehört.

Die dritte Funktion im Bunde dient dazu, einen bereits reservierten Speicherbereich in seiner Größe zu verändern. Es ist sowohl eine Vergrößerung als auch eine Verkleinerung möglich. Man übergibt beim Aufruf der Funktion 4Ah im ES-Register die Segmentadresse des in seiner Größe zu modifizierenden Speicherbereichs und im BX-Register die Anzahl der Paragraphen, die der Speicherbereich umfassen soll. Die Registerbelegung nach dem Funktionsaufruf ist mit der von Funktion 48h identisch.

Programm-Beispiel

Da sich der Aufruf der DOS-Funktionen zur Speicherverwaltung relativ einfach gestaltet und es dazu keiner besonderen Tricks bedarf, widmet sich das folgende Beispielprogramm einem anderen Thema, daß jedoch nicht minder eng mit der Speicherverwaltung des DOS zusammenhängt. Die Rede ist von einem Programm, das im System herumspioniert und ihnen alle allokierten Speicherbereiche sowie deren Inhalt anzeigt. Das Programm ist dabei intelligent genug, zwischen Speicherbereichen zu unterscheiden, die das Environment eines Programms, einen PSP oder andere Informationen enthalten.

Aufgabe dieses Programms ist es damit, sich von MCB zu MCB durch den Speicher zu "hangeln", und die allokierten Speicherbereiche zu untersuchen. Um zum jeweils nächsten MCB zu gelangen, bedient es sich dabei des dritten Felds innerhalb eines MCB, mit dessen Hilfe es einen Pointer auf den nächsten MCB erstellt. Es entsteht dadurch eine Schleife, die so lange durchlaufen wird, bis der letzte MCB entdeckt wird, dessen ID-Feld den Buchstaben "Z" enthält.

Um sich aber durch die Kette der MCBs zu bewegen, muß zunächst die Adresse der ersten Glieds der Kette, also des ersten MCB ermittelt werden. Sie verzeichnet DOS innerhalb einer internen Struktur, die den Namen DIB (DOS Information Block) trägt und für Anwendungsprogramme normalerweise nicht zugänglich ist, also ein undokumentiertes DOS-Feature darstellt. Die Adresse dieser Struktur kann jedoch mit Hilfe der Funktion 52h ermittelt werden, die diese nach ihrem Aufruf im Registerpaar ES:BX zurückgibt.

Kurioserweise zeigt die übergebene Adresse jedoch nicht auf das erste Feld innerhalb des DIB, sondern bereits auf das zweite. Da aber gerade das erste Feld die für uns wichtige Adresse des ersten MCB enthält, befindet sich die gesuchte Information hinter dem übergebenen Pointer. Da der Pointer auf den ersten MCB aus einer Offset- und Segmentadresse besteht, ist er 4 Bytes lang und dadurch an der Adresse ES:(BX-4) zu finden. Diese Adressangabe ist jedoch mit großer Vorsicht zu genießen, da sie den Anschein erweckt, man müßte lediglich den Wert 4 vom Inhalt des BX-Registers abziehen, um im Registerpaar ES:BX die effektive Adresse der gewünschten Information zu erhalten. Dies führt nur dann zum Erfolg, wenn die Offsetadresse im BX-Register größer oder gleich 4 ist. Ist sie jedoch kleiner, hat diese Vorgehensweise verherende Folgen, da sich dadurch als Ergebnis eine negative Zahl ergibt. Die jedoch gibt es bei der Speicheradressierung nicht. Um dies an einem Beispiel deutlich zu machen:

Wird im BX-Register als Offsetadresse des DIB der Wert 0 zurückgeliefert, ergibt sich durch die Subtraktion von 4 der Wert 0FFFCh. Dieser wird zwar bei arithmetischen Operationen ganz richtig als -4 interpretiert, zeigt bei Speicherzugriffen jedoch nicht auf die Adresse -4, sondern eben auf 0FFFCh und damit nicht vor, sondern ganz an das Ende des zugehörigen Segments. Klar, daß man hier die benötigte Information nicht findet.

Abhilfe schafft das Programm hier, indem es zunächst die übergebene Segmentadresse um 1 dekrementiert. Dadurch wird die effektive Adresse, die sich aus der Verknüpfung von Segment- und Offsetadresse ergibt, um den Wert 16 reduziert. Addiert man anschließend auf die Offsetadresse den Wert 12, reduziert sich die effektive Adresse gegenüber dem ursprünglichen Wert nur noch um 4 und zeigt dadurch auf die gewünschte Speicherstelle. Ohne Probleme kann dieser Speicherstelle dann die Adresse des ersten MCB entnommen werden.

Mit dieser Adresse beginnt die Schleife, in deren Verlauf alle MCBs durchlaufen und ausgewertet werden. Zunächst werden dabei einige Statusinformationen über den MCB und den von ihm verwalteten Speicherbereich ausgegeben. Dazu zählen

- die Nummer des MCB
- seine Adresse im Speicher
- die Adresse des vom MCB verwalteten Speicherbereichs
- der Inhalt des ID-Feldes ("M" oder "Z")
- die Adresse des zugehörigen PSP (unabhängig davon, ob dieser überhaupt existiert)
- die Größe des zugehörigen Speicherbereichs in Paragraphen und Bytes

Daraufhin wird der Inhalt des zugehörigen Speicherbereichs untersucht, dessen Adresse durch die Inkrementierung der Segmentadresse des MCB um den Wert 1 ermittelt wird. Zunächst wird dabei festgestellt, ob es sich bei dem Speicherbereich um einen Environment-Block handelt. Dies kann dann als sicher gelten, wenn sich am Anfang des Speicherbereichs der String COMSPEC= befindet, der jeden Environment-Block einleitet. Wird dieser String entdeckt, geht das Programm davon aus, daß es sich hier tatsächlich um einen Environment-Block handelt und gibt die einzelnen Environment-Strings aus. Diesen voran stellt es den Namen des Programms, zu dem der Environment-Block gehört, der sich ab der DOS-Version 3.0 am Ende des Environment-Blocks befindet.

Kann der Speicherbereich nicht als Environment-Block identifiziert werden, handelt es sich möglicherweise um einen PSP und damit um ein transientes oder residentes Programm. Davon geht das Programm dann aus, wenn sich in den ersten beiden Speicherstellen des Speicherbereichs der Maschinensprache-Befehl INT 20h (Code 0CDh, 020h) befindet, der jeden PSP einleitet. Trifft auch das nicht zu, kann nicht festgestellt werden, ob der Speicherbereich Programmcode, Daten oder was auch immer enthält. Um Ihnen hier die Möglichkeit zu bieten, sich ein eigenes Bild zu machen, gibt das Programm in diesem Fall die ersten 80 Bytes des Speicherbereichs als Hex- und ASCII-Dump auf dem Bildschirm aus. Nachdem der Anwender der Aufforderung, eine Taste zu betätigen, nachgekommen ist, wird dann der nächste MCB untersucht und das Programm schließlich beendet, nachdem der letzte MCB bearbeitet wurde.

```
MEMC (c) 1988 by Michael Tischer

MCB-Nummer     = 1
MCB-Adresse    = 09C8:0000
Speicher-Adr.  = 09C9:0000
ID             = M
PSP-Adresse    = 0008:0000
Größe          = 1554 Paragraphen ( 24864 Bytes )
Inhalt         = nicht identifizierbar (Programm oder Daten)

DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----+-----------------------------------------------------------------------
0000 | n p  ! , $CLOCK      6E 01 70 00 08 80 21 00 2C 00 24 43 4C 4F 43 4B
0010 |                      20 06 03 80 02 1F 1D 1F 1E 1F 1E 1F 1F 1E 1F 1E
0020 |   .    .    ╤PSQR    1F 2E 89 1E 11 00 2E 8C 06 13 00 CB 50 53 51 52
0030 | WVU      ->  & ]     57 56 55 1E 06 9C FC 0E 1F C4 3E 11 00 26 8A 5D
0040 |   √ t! √ tJ █u       02 80 FB 04 74 21 80 FB 08 74 4A 0A DB 75 03 E9
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 2
MCB-Adresse    = 0FDB:0000
Speicher-Adr.  = 0FDC:0000
ID             = M
PSP-Adresse    = 0FDC:0000
Größe          = 231 Paragraphen ( 3696 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 3
MCB-Adresse    = 10C3:0000
Speicher-Adr.  = 10C4:0000
ID             = M
PSP-Adresse    = 0000:0000
Größe          = 3 Paragraphen ( 48 Bytes )
Inhalt         = nicht identifizierbar (Programm oder Daten)

DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----+-----------------------------------------------------------------------
0000 |        ╦             00 01 00 00 00 00 00 CB 00 00 00 FF FF FF FF FF
0010 |               C      FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 43
0020 | :\AUTOEXEC.BAT       3A 5C 41 55 54 4F 45 58 45 43 2E 42 41 54 00 00
0030 | M                    4D DC 0F 0A 00 00 00 00 00 00 00 00 00 00 00 00
0040 | COMSPEC=C:\COMMA     43 4F 4D 53 50 45 43 3D 43 3A 5C 43 4F 4D 4D 41
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 4
MCB-Adresse    = 10C7:0000
Speicher-Adr.  = 10C8:0000
ID             = M
PSP-Adresse    = 0FDC:0000
Größe          = 10 Paragraphen ( 160 Bytes )
Inhalt         = Environment
Programmname   = unbekannt
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 5
MCB-Adresse    = 10D2:0000
Speicher-Adr.  = 10D3:0000
ID             = M
PSP-Adresse    = 10DD:0000
Größe          = 9 Paragraphen ( 144 Bytes )
Inhalt         = Environment
Programmname   = C:\DOS\KEYB.COM
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 6
MCB-Adresse    = 10DC:0000
Speicher-Adr.  = 10DD:0000
ID             = M
PSP-Adresse    = 10DD:0000
Größe          = 341 Paragraphen ( 5456 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 7
MCB-Adresse    = 1232:0000
Speicher-Adr.  = 1233:0000
ID             = M
PSP-Adresse    = 123D:0000
Größe          = 9 Paragraphen ( 144 Bytes )
Inhalt         = Environment
Programmname   = C:\DOS\CED.COM
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 8
MCB-Adresse    = 123C:0000
Speicher-Adr.  = 123D:0000
ID             = M
PSP-Adresse    = 123D:0000
Größe          = 1030 Paragraphen ( 16480 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 9
MCB-Adresse    = 1643:0000
Speicher-Adr.  = 1644:0000
ID             = M
PSP-Adresse    = 164E:0000
Größe          = 9 Paragraphen ( 144 Bytes )
Inhalt         = Environment
Programmname   = C:\DOS\CACHE-AT.COM
Environment-Strings
```

Abbildung 3: *Die Ausgabe des Programms MEMC.C*

```
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 10
MCB-Adresse    = 164D:0000
Speicher-Adr.  = 164E:0000
ID             = M
PSP-Adresse    = 164E:0000
Größe          = 1922 Paragraphen ( 30752 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 11
MCB-Adresse    = 1DD0:0000
Speicher-Adr.  = 1DD1:0000
ID             = M
PSP-Adresse    = 1DDC:0000
Größe          = 10 Paragraphen ( 160 Bytes )
Inhalt         = Environment
Programmname   = C:\DOS\KEYBUF.COM
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 12
MCB-Adresse    = 1DDB:0000
Speicher-Adr.  = 1DDC:0000
ID             = M
PSP-Adresse    = 1DDC:0000
Größe          = 27 Paragraphen ( 432 Bytes )
Inhalt         = nicht identifizierbar (Programm oder Daten)
DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----+-----------------------------------------------------------------------
0000 |  M M M M M M M M     00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D
0010 |  M M M M M M M M     00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D
0020 | + 1  K K K K K K     2B 1B 31 02 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
0030 |  K K K K K K K K     00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
0040 |  K K K K K K K K     00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 13
MCB-Adresse    = 1DF7:0000
Speicher-Adr.  = 1DF8:0000
ID             = M
PSP-Adresse    = 0FDC:0000
Größe          = 4 Paragraphen ( 64 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 14
MCB-Adresse    = 1DFC:0000
Speicher-Adr.  = 1DFD:0000
ID             = M
PSP-Adresse    = 1E08:0000
Größe          = 10 Paragraphen ( 160 Bytes )
Inhalt         = Environment
Programmname   = D:\PCI\C\QC.EXE
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 15
MCB-Adresse    = 1E07:0000
Speicher-Adr.  = 1E08:0000
ID             = M
PSP-Adresse    = 1E08:0000
Größe          = 16200 Paragraphen ( 259200 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 16
MCB-Adresse    = 5D50:0000
Speicher-Adr.  = 5D51:0000
ID             = M
PSP-Adresse    = 5D5C:0000
Größe          = 10 Paragraphen ( 160 Bytes )
Inhalt         = Environment
Programmname   = C:\QC\OBEX\MEMDEMO.EXE
Environment-Strings
           COMSPEC=C:\COMMAND.COM
           PATH=C:\;C:\DOS;C:\BATCHES;E:\;D:\MSC\BIN
           INCLUDE=d:\msc\include
           LIB=d:\msc\lib
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 17
MCB-Adresse    = 5D5B:0000
Speicher-Adr.  = 5D5C:0000
ID             = M
PSP-Adresse    = 5D5C:0000
Größe          = 4512 Paragraphen ( 72192 Bytes )
Inhalt         = PSP (mit nachfolgendem Programm)
======================================================== Bitte Taste drücken ==
MCB-Nummer     = 18
MCB-Adresse    = 6EFC:0000
Speicher-Adr.  = 6EFD:0000
ID             = Z
PSP-Adresse    = 0000:0000
Größe          = 12547 Paragraphen ( 200752 Bytes )
Inhalt         = nicht identifizierbar (Programm oder Daten)
DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----+-----------------------------------------------------------------------
0000 | H   6   %     .      48 00 00 00 36 00 08 00 25 00 88 08 00 00 A8 00
0010 | (   v         '      28 04 1E 00 76 00 17 00 FF FF 96 08 00 00 27 0C
0020 | H   Q         .      48 1F 00 00 51 00 1E 00 FF FF 96 08 90 01 2E 05
0030 | HB  6   %     B      48 42 00 00 36 00 08 00 25 00 96 08 EC 01 42 00
0040 | (   v         E      28 0D E3 00 76 00 17 00 FF FF A4 08 00 00 45 0C
======================================================== Bitte Taste drücken ==
```

Abbildung 3: *(Ende)*

```c
/*****************************************************************************/
/*                               M E M C                                     */
/*---------------------------------------------------------------------------*/
/*    Aufgabe        : Verfolgt die Kette der über DOS allokierten           */
/*                     Speicherblöcke.                                       */
/*---------------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                       */
/*    entwickelt am  : 23.08.1988                                            */
/*    letztes Update : 23.08.1988                                            */
/*---------------------------------------------------------------------------*/
/*    Erstellung     : CL /AS /Zp memc.c                                     */
/*    Aufruf         : MEMC                                                  */
/*****************************************************************************/

/*== Include-Dateien einbinden ==============================================*/

#include <dos.h>
#include <stdlib.h>

/*== Typedefs ===============================================================*/

typedef unsigned char byte;                      /* wir basteln uns ein Byte */
typedef unsigned segadr;                           /* eine Segmentadresse */
typedef byte boolean;
typedef byte far *FB;                            /* FAR-Pointer auf ein Byte */

/*== Konstanten =============================================================*/

#define TRUE  1              /* werden zur Arbeit mit BOOLEAN benötigt */
#define FALSE 0

/*== Strukturen und Unions ==================================================*/

struct MCB {                              /* beschreibt einen MCB im Speicher */
            byte     id_code; /* 'M' = es folgt ein Block, 'Z' = Ende */
            segadr   psp;        /* Segmentadresse des zugehörigen PSP */
            unsigned abstand;  /* Anzahl der reservierten Paragraphen */
           };

typedef struct MCB far *MCBPtr;                /* FAR Pointer auf einen MCB */

/*== Makros =================================================================*/

#ifndef MK_FP                         /* wurde MK_FP noch nicht definiert? */
#define MK_FP(seg, ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
#endif

/*****************************************************************************
* Funktion          : F I R S T _ M C B                                      *
**--------------------------------------------------------------------------**
* Aufgabe           : Liefert einen Pointer auf den ersten MCB.              *
* Eingabe-Parameter: keine                                                   *
* Return-Wert       : Pointer auf den ersten MCB                             *
*****************************************************************************/

MCBPtr first_mcb()
{
 union REGS regs;                        /* nimmt die Prozessorregister auf */
 struct SREGS sregs;                       /* nimmt die Segmentregister auf */

 regs.h.ah = 0x52;       /* Fkt.nr.: Adresse des DOS-Info-Block holen */
 intdosx( &regs, &regs, &sregs );         /* DOS-Interrupt 0x21 aufrufen */

 /*-- ES:(BX-4) zeigt auf den ersten MCB, Pointer erstellen ----------*/

 return( *((MCBPtr far *) MK_FP( sregs.es-1, regs.x.bx+12 )) );
}

/*****************************************************************************
* Funktion          : D U M P                                                *
**--------------------------------------------------------------------------**
* Aufgabe           : Gibt Hex- und ASCII-Dump eines Speicherbereiches       *
*                     aus.                                                   *
* Eingabe-Parameter: - bptr : Pointer auf den Speicherbereich                *
*                    - anz  : Anzahl der Dumpzeilen (je 16 Byte)             *
* Return-Wert       : keiner                                                 *
*****************************************************************************/

void dump( FB bptr, byte anz)
{
 FB lptr;                   /* Laufzeiger zur Ausgabe einer Dump-Zeile */
 unsigned offset;                      /* Offsetadresse relativ zu BPTR */
 byte i;                                              /* Schleifenzähler */

 printf("\nDUMP | 0123456789ABCDEF        00 01 02 03 04 05 06 07 08");
 printf(" 09 0A 0B 0C 0D 0E 0F\n");
 printf("─────┼──────────────────────────────────────────────────");
 printf("──────────────────\n");

 for (offset=0;  anz-- ; offset += 16, bptr += 16)
  {                                   /* die Schleife ANZ mal durchlaufen */
   printf("%04x | ", offset);
   for (lptr=bptr, i=16; i-- ; ++lptr)  /* Zeichen als ASCII ausgeben */
    printf("%c", (*lptr<32) ? ' ' : *lptr);
   printf("       ");
   for (lptr=bptr, i=16; i-- ; )            /* Zeichen als Hex ausgeben */
    printf("%02X ", *lptr++);
   printf("\n");                        /* in die nächste Zeile schalten */
  }
}
```

Listing 1: *MEMC.C*

```c
/*****************************************************************************
* Funktion          : T R A C E _ M C B                                      *
**--------------------------------------------------------------------------**
* Aufgabe           : Verfolgt die Kette der MCBs.                           *
* Eingabe-Parameter: keine                                                   *
* Return-Wert       : keine                                                  *
*****************************************************************************/

void trace_mcb()
{
 static char fenv[] = {                      /* erster Environment-String */
                      'C', 'O', 'M', 'S', 'P', 'E', 'C', '='
                     };
 MCBPtr akt_mcb;               /* Pointer auf den jeweils aktuellen MCB */
 boolean ende;          /* wird TRUE, wenn der letzte MCB gefunden ist */
 byte nr_mcb,                  /* Nummer des aktuell bearbeiteten MCB */
      i;                                          /* Schleifenvariable */
 FB lptr;                              /* Laufzeiger in das Environment */

 ende = FALSE;                                /* jetzt geht's erst mal los */
 nr_mcb = 1;                        /* der erste MCB trägt die Nummer 1 */
 akt_mcb = first_mcb();             /* Pointer auf den ersten MCB holen */
 do                                      /* die einzelnen MCBs abarbeiten */
  {
   if ( akt_mcb->id_code == 'Z' )                /* letzten MCB erreicht? */
    ende = TRUE;                                                  /* Ja */
   printf("MCB-Nummer     = %d\n", nr_mcb++);
   printf("MCB-Adresse    = %Fp\n", akt_mcb);
   printf("Speicher-Adr.  = %Np:0000\n", FP_SEG(akt_mcb)+1);
   printf("ID             = %c\n", akt_mcb->id_code);
   printf("PSP-Adresse    = %Fp\n", (FB) MK_FP(akt_mcb->psp, 0) );
   printf("Größe          = %u Paragraphen ( %lu Bytes )\n",
          akt_mcb->abstand, (unsigned long) akt_mcb->abstand << 4);
   printf("Inhalt         = ");

   /*-- handelt es sich um ein Environment? ---------------------------*/
   for (i=0, lptr=(FB)akt_mcb+16; /* ersten ENV-String mit FENV vergl. */
        ( i<sizeof fenv ) && ( *(lptr++) == fenv[i++] ) ; )
    ;
   if ( i == sizeof fenv )                  /* wurde der String entdeckt? */
    {                       /* Ja, es handelt sich um ein Environment */
     printf("Environment\n");
     if ( _osmajor >= 3 )                 /* DOS-Version 3.0 oder höher? */
      {                                  /* Ja, Programmnamen ermitteln */
       printf("Programmname  = ");
       for ( ; !(*(lptr++)==0 && *lptr==0) ; )
        ;                                   /* letzten ENV-String suchen */
       if ( *(lptr += 3) )      /* befindet sich hier ein Prgrammname? */
        {                                                          /* Ja */
         for ( ; *lptr ; )              /* den Programmnamen durchlaufen */
          printf( "%c", *(lptr++) );  /* jeweils ein Zeichen ausgeben */
        }
       else                          /* es wurde kein Programmname entdeckt */
        printf("unbekannt");
       printf("\n");                    /* in die nächste Zeile schalten */
      }

     /*-- die Environment-Strings ausgeben ---------------------------*/
     printf("Environment-Strings\n");
     for (lptr=(FB) akt_mcb +16; *lptr ; ++lptr)
      {                                          /* einen String ausgeben */
       printf("              ");
       for ( ; *lptr ; ) /* den String bis zum NUL-Zeichen durchlaufen */
        printf( "%c", *(lptr++) );     /* jeweils ein Zeichen ausgeben */
       printf("\n");                    /* in die nächste Zeile schalten */
      }
    }
   else                                               /* kein Envrionment */
    {
     /*- handelt es sich um einen PSP? ---------------------------------*/
     /*-- (wird durch Befehl INT 20 (Code=0xCD 0x20) eingeleitet) -----*/
     if (*((unsigned far *) MK_FP( akt_mcb->psp, 0 )) == 0x20cd)
      printf("PSP (mit nachfolgendem Programm)\n");                /* Ja */
     else         /* der Befehl INT 0x20 konnte nicht entdeckt werden */
      {
       printf("nicht identifizierbar (Programm oder Daten)\n");
       dump( (FB) akt_mcb + 16, 5);        /* die ersten 5*16 Bytes dumpen */
      }
    }

   printf("════════════════════════════════════════════════");
   printf("═════════════════ Bitte Taste drücken ══\n");
   if ( !ende )                                    /* folgt noch ein MCB ? */
    {                        /* Ja, Pointer auf den nächsten MCB setzen */
     akt_mcb = (MCBPtr)
               MK_FP( FP_SEG(akt_mcb) + akt_mcb->abstand + 1, 0 );
     getch();                                    /* auf eine Taste warten */
    }
  }
 while ( !ende );  /* wiederholen, bis der letzte MCB bearbeitet ist */
}

/*****************************************************************************/
/**                            HAUPTPROGRAMM                                **/
/*****************************************************************************/
void main()
{
 printf("\nMEMC (c) 1988 by Michael Tischer\n\n");
 trace_mcb();                           /* die Kette der MCB nachverfolgen */
}
```

Listing 1: *(Ende)*

Um Ihnen eine Interpretationshilfe bei der Auswertung der Ausgaben des Programms zu geben, zeigt die *Abbildung 3* die Ausgaben des Programms, nachdem es auf dem Rechner des Autors gestartet wurde. In den folgenden Absätzen wird die Bedeutung der einzelnen Speicherbereiche beschrieben.

1 Der erste MCB kann zwar vom Programm nicht identifiziert werden (die angegebene PSP-Adresse besitzt daher keinerlei Informationswert), doch läßt der beigefügte Speicherauszug seinen Inhalt erahnen. In der ersten Zeile des ASCII-Dump findet sich das Wort $CLOCK, der Name, mit dem DOS den Gerätetreiber für die interne Uhr bezeichnet. Und tatsächlich scheint es sich hier um den Speicher für einen Gerätetreiber zu handeln, denn der Aufbau der ersten 18 Byte entspricht genau dem Aufbau des Kopfes eines solchen Geräte-Treibers. Dabei kann es sich jedoch nicht um einen der fest installierten Geräte-Treiber des DOS handeln, da sie unterhalb der TPA (Transient Program Area) installiert werden und für sie deshalb kein Speicher allokiert werden muß. Darum muß es sich hier um einen Geräte-Treiber handeln, der beim Booten des Systems über den DEVICE-Befehl innerhalb der Konfigurationsdatei CONFIG.SYS installiert wird. Tatsächlich habe ich dort als ersten Geräte-Treiber den Treiber AT-UHR.SYS aufgeführt, der als Geräte-Namen den Namen $CLOCK trägt. Um diesen Treiber muß es sich hier also handeln.
Da dieser Treiber jedoch nur wenige KByte groß ist, der allokierte Speicherbereich darüber jedoch weit hinaus geht, muß sich hinter dem Treiber noch anderer Programmcode oder Daten befinden. Ein Blick über die 5 Zeilen des Dump, wie sie das Programm ausgibt, hinaus zeigt, daß dahinter auch alle anderen Treiber folgen, die ich über den DEVICE-Befehl eingebunden habe. Der erste allokierte Speicherbereich wird vom DOS also bereits während des Bootvorgangs allokiert, um die einzubindenden Geräte-Treiber in der Reihenfolge ihrer Nennung innerhalb der Konfigurationsdatei aufzunehmen.

2 In diesem Speicherbereich befindet sich offensichtlich ein Programm. Da ihm jedoch kein PSP vorangeht, der seinen Namen verraten könnte, bleibt der Name des Programms unbekannt. An seiner Lage innerhalb der MCB-Kette und innerhalb des Speichers kann man jedoch feststellen, daß es bereits kurz nach dem Bootvorgang in den Speicher gebracht und dort resident installiert wurde.

3 Über den Inhalt dieses Speicherbereichs kann keine Aussage getroffen werden. Entweder ist er von einem Programm allokiert worden, um später Daten darin abzulegen, oder er ist einfach bei der Freigabe von Speicher "übrig geblieben".

4 Ganz offensichtlich handelt es sich hier um ein Environment, dem jedoch der zugehörige Programmname fehlt. Als Adresse des PSP wird die Adresse 0FDC:0000 angegeben, an der sich der Speicherbereich befindet, der über den MCB 2 allokiert wurde. Da es sich bei MCB 2 um einen PSP und bei MCB 4 um ein Environment handelt, liegt die Vermutung nahe, daß es sich hier um ein Programm und dessen zugehöriges Environment handelt. Da der Environment-Block keinen Programmnamen enthält, kann davon ausgegangen werden, daß das Programm im MCB 2 weder über die Benutzeroberfläche des DOS noch durch einen Befehl innerhalb einer BATCH-Datei eingeladen und gestartet wurde. Vielmehr scheint es sich beim MCB 2 um den residenten Teil des Kommandprozessors COMMAND.COM zu handeln, der sein Environment im MCB 4 abgelegt hat. Untersucht man den Programmcode im MCB 2 mit Hilfe eines Debuggers bestätigt sich diese Vermutung.

5 Hier befindet sich das Environment des Programms KEYB.COM, das die Arbeit mit der deutschen Tastatur erlaubt und innerhalb meiner AUTOEXEC.BAT durch den Befehl KEYB GR gestartet wird. Da es sich noch immer im Speicher befindet, handelt es sich bei ihm um ein residentes Programm, das nach seiner Installation im Speicher verbleibt.

6 Während sich das Environment des Programms KEYB. COM im MCB 5 befindet, haben wir hier den Speicher für dieses Programm (also PSP, Programmcode und Daten) vor uns. Erkennbar wird dies bereits daran, daß die PSP-Adresse im Environment-Block (MCB 5) auf den Speicherbereich zeigt, der durch diesen MCB verwaltet wird.

7, 8 Diese beiden Speicherbereiche nehmen das Environment und den PSP (bzw. den Programmcode) des Programms CED.COM auf, das ich innerhalb der AUTOEXEC.BAT aufrufe. Da es sich noch immer im Speicher befindet, handelt es sich bei ihm offensichtlich um ein residentes Programm.

9, 10 Wie 7 und 8, jedoch für CACHE-AT.COM.

11, 12 Auch hier handelt es sich um die Speicherbereiche eines Programms und seines Environments. Zwar ist das Environment eindeutig zu erkennen, doch konnte der PSP nicht erkannt werden. Dies liegt daran, daß dieses Programm seinen PSP als Tastaturpuffer mißbraucht und daher der Interrupt-Aufruf am Anfang des PSP überschrieben wird. Da dieser Befehl als PSP-Kennung dient, kann dieser Speicherbereich nicht mehr als PSP erkannt werden.

13 Zwar meldet das Programm, daß es sich hier um einen PSP handelt, doch kann es sich hier lediglich um den Anfang eines PSP handeln, da der Speicherbereich nur 64 Byte groß ist, ein PSP aber 256 Byte benötigt. Wahrscheinlich befand sich an dieser Stelle einmal ein PSP, der aber nicht komplett wieder freigegeben wurde, so daß sich ein Rest immer noch im Speicher befindet.

14, 15 Da das Programm zur Ausgabe der MCBs und ihrer Inhalte in C verfaßt wurde, befand ich mich innerhalb

der QuickC-Umgebung als ich das Programm startete, um den Speicherauszug zu erstellen, von dem ich gerade spreche. In den MCBs 14 und 15 findet sich deshalb das Environment und der Programmcode des QuickC-Programms.

16, 17 Um einen MCB- bzw. Speicherdump zu erhalten, habe ich das Programm MEMDEMO innerhalb von QuickC zur Ausführung gebracht. Die Ausführung des Programms hat QuickC dadurch gestartet, daß es das Programm mit Hilfe des EXEC-Loaders als Tochter-Prozeß aufgerufen hat. Die Speicherbereiche 16 und 17 sind daher vom EXEC-Loader allokiert worden, um das Programm auszuführen. Nach Programm-Beendigung dürften sie wieder freigegeben worden sein.

18 Der letzte Speicherblock umfaßt den Rest des Speichers, der noch nicht allokiert wurde. In diesem konkreten Fall sind das knapp 200 KByte.

Das C-Beispielprogramm in *Listing 1* gibt einen MCB-Dump aus, wie er in *Abbildung 1* gezeigt wurde.

Das Programm arbeitet zum Zugriff auf den Speicher mit FAR-Pointern, da sich die zu adressierenden Speicherbereiche außerhalb seines Datensegments befinden. Daß die Pointer vom Typ FAR sind, wird innerhalb von C durch die Wahl eines entsprechenden Speichermodells (Compact, Huge oder Large) oder mit Hilfe von Cast-Operatoren erreicht, die jeweils explizit die Arbeit mit einem FAR-Pointer definieren. Der letztere Weg wurde bei diesem Programm beschritten, um auch die Kompilierung unter einem Speichermodell zu erlauben, das standardmäßig mit NEAR-Pointern arbeitet (Small und Medium).

Ein Problem stellt die Umwandlung einer getrennt ermittelten Offset- und Segmentadresse in einen FAR-Pointer dar. In C kann dies mit einem Makro realisiert werden. Über diese kurzen Anmerkungen hinaus sollte das Listing für sich sprechen können, da es vollständig dokumentiert ist.

Michael Tischer

OS/2-Programme unter DOS:

Presentation Manager-Bibliothek für DOS

Mit dem Entwicklungstool QuickStep Presentation Manager stellt Lauer & Wallwitz einen zum OS/2 Presentation Manager von Microsoft kompatiblen Window-Manager zur Anwendung unter DOS und UNIX vor. Der QuickStep Presentation Manager arbeitet vollständig im Textmodus und schließt die Lücke zwischen DOS und OS/2. Rainer Wallwitz beschreibt hier die Ideen und Konzepte, die hinter dem QuickStep Presentation Manager stehen.

Die Idee

Den ersten Kontakt zu MS-Windows hatten wir 1986/87 anläßlich der Entwicklung zweier neuer Applikationen. Obwohl die Entscheidung, Windows als Benutzeroberfläche einzusetzen, zum damaligen Zeitpunkt sicher exotisch genannt werden kann - es gab noch keine Tools und fast keine Informationen - erschien sie uns in Hinblick auf die Absicht Microsofts, auch OS/2 mit einer grafischen Oberfläche zu versehen, als sinnvoll. Nun, zwei Jahre später, ist diese grafische Oberfläche, der OS/2 Presentation Manager für OS/2 und Windows in der Version 2.x für DOS, verfügbar. So weit so gut. Vollkommen zufriedenstellend ist die Situation jedoch leider noch nicht, wie wir bei unseren ersten Gehversuchen mit dem Presentation Manager SDK feststellen mußten. Der Presentation Manager stellt eine erhebliche Weiterentwicklung und Verbesserung des Windows-Konzeptes auf Kosten der Kompatibilität zwischen beiden dar.

Obwohl diese Weiterentwicklung zu begrüßen war, standen wir als Entwickler von MS-Windows-Applikationen vor dem Problem, daß einerseits unser Quellcode aufgrund der Inkompatibilitäten zwischen Windows und dem Presentation Manager nur mit erheblichem Aufwand nach OS/2 zu portieren ist und andererseits nach einer erfolgten Portierung zwei Versionen der Programme existieren würden, die getrennt gewartet und weiterentwickelt werden müßten. Hinzu kommt, daß wir der Grenze an freiem Speicherplatz immer näher kamen und das unsere Applikationen bzw. MS-Windows eine gute grafische Hardware voraussetzen. Dies mag unter DOS nur eingeschränkt gelten, für UNIX jedoch, wo die Mehrzahl der Arbeitsplätze mit ASCII-Terminals ausgestattet ist, sind Windows- oder Presentation Manager-Anwendungen überhaupt nicht einsetzbar.

Die Lösung war offensichtlich: Man braucht zusätzlich zu der grafischen Oberfläche Microsofts unter OS/2 und UNIX (zur Zeit in Entwicklung) eine Oberfläche unter DOS, die im Textmodus arbeitet, weitestgehgend kompatibel zum OS/2 Presentation Manager ist und die unter DOS nicht mehr als 100 KByte Speicherplatz benötigt. Die Oberfläche sollte sich dabei in einer Bibliothek befinden, deren Funktionen nach Bedarf mit denen der Applikation zusam-

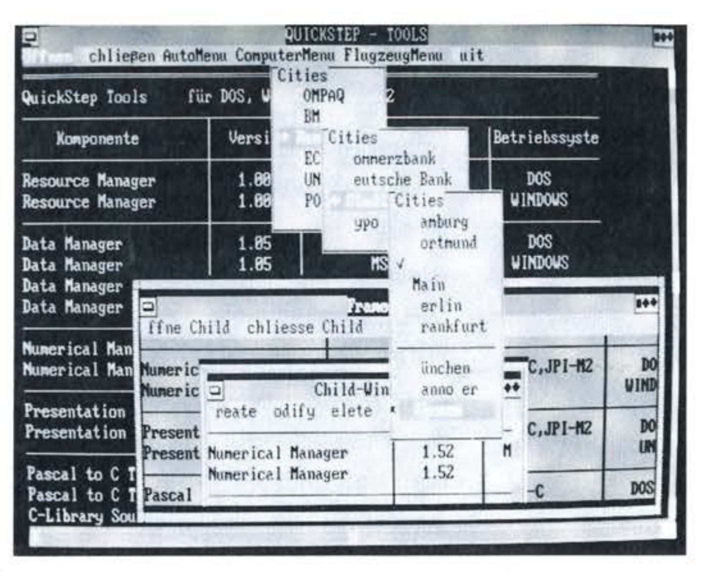


Abbildung 1: *So sieht ein mit dem QuickStep Presentation Manager geschriebenes Programm unter DOS aus.*

mengelinkt werden können. Im Fall der Verfügbarkeit von ausreichend guter Grafik-Hardware unter OS/2 und UNIX linkt man mit den Bibliotheken von Microsoft, ansonsten, vor allem unter DOS, mit den QuickStep-Bibliotheken. Auf diese Weise erreicht man einen hohen Grad an Portabilität zwischen den Betriebssystemen DOS, OS/2 und UNIX und kann vor allem mit der zur Zeit vorhandenen Hardware (640 KByte, Real-Mode) und seinen individuellen Entwicklungstools Applikationen für den Presentation Manager unter DOS entwickeln.

Im Laufe des Jahres 1987 beschlossen wir das Projekt Presentation Manager/DOS anzugehen und eine entsprechende Bibliothek zu entwickeln, die auf der CeBIT 89 vorgestellt werden sollte. Wir liegen zwar recht gut in der Zeit, aber dennoch kann hier nur eine Vorversion beschrieben und vorgestellt werden, obwohl fast alle Features, von Dialogboxen und Clipboardfunktionen abgesehen, implementiert sind. Als Implementationssprache der DOS-Version wurde Microsoft-C (Version 5.1) in Kombination mit dem Microsoft Macro-Assembler Version 5.1 benutzt, wobei nur die Hardwaretreiber (Maus, Tastatur, Timer, Video) aus Platz- und Performancegründen in Assembler geschrieben sind.

Die entwickelte Presentation Manager-Bibliothek für DOS ist ein Teil unserer QuickStep-Tools, einer Bibliotheksserie für Entwicklungen mit C, Pascal und Modula-2 unter DOS, Windows, OS/2 und UNIX. Der QuickStep Presentation Manager (QS-PM) ist von seinem Funktionsumfang und »look & feel« im wesentlichen kompatibel zum Microsoft OS/2 Presentation Manager (OS/2-PM). Im Gegensatz zum OS/2-PM ist der QS-PM jedoch keine Betriebssystemerweiterung, sondern eine Bibliothek, die vom Programmierer in seine Applikation eingebunden werden muß, vergleichbar den Dynamic-Link-Bibliotheken von Microsoft.

Funktionsübersicht der OS/2 PM kompatiblen Funktionen	Tab.1
GpiCharStringAt	Ausgabe eines Strings
WinBeginPaint	Einleitung der WM_PAINT-Nachricht
WinCreateWindow	Erzeugung eines Windows
WinCreateCursor	Erzeugung eines Tastatur-Cursors
WinCreateMsgQueue	Erzeugung der System-Queue
WinCreateStdWindow	Erzeugung eines Standard- bzw. Frame-Windows
WinDestroyCursor	Zerstörung des Tastatur-Cursors
WinDefWindowProc	Default-Windowfunktion
WinDestroyWindow	Zerstörung eines Windows
WinDispatchMsg	Verteilen der Nachrichten
WinDestroyMsgQueue	Zerstörung der System-Queue
WinEndPaint	Ende der WM_PAINT-Nachricht
WinFillRect	Rechteck füllen
WinGetMsg	Message aus System-Queue lesen
WinGetKeyState	Tastaturstatus lesen
WinGetPS	Presentation Space holen
WinInvalidateRect	Rechteck invalidieren
WinInitialize	Presentation Manager initialisieren
WinIntersectRect	Schnittrechteck berechnen
WinLoadString	String aus Ressouce laden
WinLoadAccelTable	Accelerator-Tabelle aus Ressource laden
WinLoadMenu	Menu aus Ressource laden
WinMessageBox	Message-Box darstellen
WinQueryFocus	Input-Focus erfragen
WinQueryWindow	Window-Handle mit QW_* erfragen
WinPostMsg	Message in die System-Queue legen
WinPeekMsg	Message aus System-Queue lesen
WinPtInRect	Punkttest
WinQueryWindowRect	Rechteck des Windows erfragen
WinQueryActiveWindow	aktive Window-Handle erfragen
WinQueryPointerPos	Position des Mauszeigers erfragen
WinQueryWindowText	Text der Titelbar-Control eines Windows erfragen
WinQueryWindowULong	32-Bitwert aus lokalem Windowbereich lesen
WinRestrictPointer	Mauszeiger in Rechteck einschließen
WinRegisterClass	Windowklasse registrieren
WinSendMsg	Message versenden
WinSetWindowULong	32-Bitwert in lokalen Windowbereich schreiben
WinSetWindowText	Text der Titelbar-Control eines Windows setzen
WinSetPointer	Mauszeiger erzeugen
WinSetFocus	Focus-Window festlegen
WinSetRect	Rechteck definieren
WinSetPointerPos	Mauszeiger positionieren
WinSetCapture	Maus-Message abfangen
WinSetActiveWindow	aktives Window definieren
WinSetWindowPos	Setzen der Windowposition
WinShowCursor	Tastaturcursor darstellen bzw. verstecken
WinShowPointer	Mauszeiger darstellen bzw. verstecken
WinShowWindow	Window darstellen bzw. verstecken
WinStartTimer	Timerprozeß starten
WinStopTimer	Timerprozeß anhalten
WinSubclassWindow	Window-Subclassing
WinTerminate	Presentation Manager beenden
WinUpdateWindow	Windowinhalt aktualisieren
WinUnionRect	Vereinigung zweier Recheckflächen
WinWindowFromID	Window-Handle aus FID_*-Werten
WinWindowFromPoint	Window-Handle aus Positionsinformation
Zusätzliche Funktionen (Auszug)	
WinCustomizeQS	optionale Konfiguration der Treiber
WinLoadStringPtrQS	liefert Zeiger auf String-Ressource
WinQueryTextColorQS	aktuelle Zeichenfarbe von Window erfragen
WinQueryViewportOrgQS	aktuelle Position des Viewports erfragen
WinQueryViewportExtQS	aktuelle Ausdehnung des Viewports erfragen
WinQueryWindowOrgQS	aktuelle Position des Windows erfragen
WinQueryWindowExtQS	aktuelle Ausdehnung des Windows erfragen
WinSetTextColorQS	aktuelle Zeichenfarbe des Windows setzen
WinSetWindowOrgQS	logische Position des Windows setzen
WinSetWindowExtQS	logische Länge des Window setzen
WinSetViewportOrgQS	logische Position des Viewports setzen
WinSetViewportExtQS	logische Länge des Viewports setzen

Tabelle 1: *Funktionsübersicht der OS/2-PM-kompatiblen Funktionen*

Die Implementation

Der QS-PM ist, analog dem OS/2-PM, Message- bzw. Event-Driven und verfügt über ein identisches Maus-, Timer- und Tastaturinterface. Des weiteren sind die meisten der Nachrichten wie Window- (WM_*), Scrollbar- (SBM_*), Titlebar- (TBM_*) und Menu- (MM_*) Nachrichten und Funktionen implementiert. Die Tabellen *1 bis 3* zeigen die wichtigsten unterstützten Funktionen (*Tabelle 1*), Datenstrukturen (*Tabelle 2*) und Nachrichten (*Tabelle 3*). Die größten Unterschiede bestehen naturgemäß in der Art der Textdarstellung.

Datenstrukturen		Tab.2
POINT	Positionsangabe (x,y)	16-Bit
RECT	Rechteckdefinition	16-Bit
POINTL	Positionsangabe (x,y)	32-Bit
RECTL	Rechteckdefinition	32-Bit
QMSG	Nachrichtenstruktur	
WNDCLASS	Daten der Windowklasse	
CREATESTRUCT	enthält Informationen zur Kreierungszeit	
TRACKINFO	Information über das aktuelle Tracking-Rectangle	
MENUITEM	Struktur eines Menueintrages	
WNDPARAMS	Daten des Windows	
CLASSINFO	Informationen über die Windowklasse	

Tabelle 2: *Datenstrukturen*

Die im grafischen Modus (OS/2-PM) notwendigen Koordinatentransformationen vom Window- in das Viewportkoordinatensystem gestalten sich im Textmodus einfacher, da hier beide Koordinatensysteme die gleiche Skalierung besitzen und nur ihr Ursprung gegeneinander verschoben sein kann. Die mit der Textdarstellung verknüpfte Farbinformation wird im QS-PM als Attributbyte verwaltet, wohingegen im OS/2-PM eine 32-Bit-Größe zur Kodierung der verschiedenen Farbebenen benutzt wird. Vektorfonts wird der QS-PM-Programmierer gänzlich vermissen, da sie im Textmodus nicht implementierbar sind. Raster- bzw. Bitmapfonts dagegen werden unterstützt, sofern die jeweilige Videohardware dies zuläßt. Die *Abbildung 1* ist zum Beispiel an einem EGA-Bildschirm gemacht worden, bei dem ein Teil des Zeichensatzes durch die Funktionen des entsprechenden Videotreibers zur Angleichung an die grafische Variante des PMs umdefiniert wurde.

Die Darstellungsgeschwindigkeit des QS-PM läßt nichts zu wünschen übrig, da der Videotreiber direkt auf den Bildschirmspeicher zugreift. Der Platzbedarf des QS-PM-Kernels liegt etwa zwischen 60 - 80 KByte. Das aufgelistete Beispielprogramm DEMO.C (*Listing 1*) besitzt als EXE-Datei eine Größe von ca. 91 Kbyte, wobei in diesem Programm der größte Teil der Laufzeitbibliothek und zahlreiche String- und Menüdefinitionen enthalten sind. Ähnlich der Konvention unter MS-Windows und OS/2-PM, wird im QS-PM die Pascal-Aufrufsequenz und als Speichermodell das Large-Model benutzt, so daß alle Datenzeiger 32-Bit lang sind. Die Benutzung der Pascal-Aufrufsequenz hat zwei Vorteile: Erstens verkürzt sich der Codeumfang des Kernels um ca. 4 KByte und zweitens ist die Pascal-Callingsequenz besser zur Integration von in Pascal, Modula-2 und insbesondere von in Assembler geschriebenen Routinen geeignet.

Die gesamte Textausgabe seitens des Entwicklers findet über die beiden Funktionen GpiCharStringAt() und GpiTextOutQS() statt, von denen die erste eine OS/2-PM-kompatible Funktion ist und auf die zweite zugreift. Die Darstellung findet immer in der aktuellen Farbe und in dem Window statt, dessen Presentation-Space Handle (hPS) beim Aufruf von GpiCharStringAt() übergeben wurde. Die Farbunterstützung unterscheidet sich jedoch etwas von der

Allgemeine Nachrichten	Tab. 3
WM_NULL	die NULL-Nachricht
WM_CREATE	wird nach der Kreierung gesendet
WM_DESTROY	wird vor der Zerstörung gesendet
WM_MOVE	bei Veränderung der Position gesendet
WM_SIZE	bei Veränderung der Größe gesendet
WM_ACTIVATE	bei der Aktivierung/Deaktivierung gesendet
WM_SETFOCUS	beim Erhalt/Verlust des Input-Focus gesendet
WM_PAINT	zur Aktualisierung gesendet
WM_CLOSE	vor dem Schließen gesendet
WM_SHOW	vor dem Öffnen gesendet
WM_QUIT	dient zum Beenden der Applikation
WM_SETWINDOWPARAMS	Windowparameter setzen
WM_QUERYWINDOWPARAMS	Windowparameter erfragen
WM_FLASHWINDOW	dient zum "flashing" des Rahmens
WM_ERASEBACKGROUND	dient zum Löschen des Windowhintergrungs
WM_CHAR	beim Drücken einer Taste gesendet
WM_COMMAND	bei der Anwahl eines Menupunktes oder Accelerator gesendet
WM_SYSCOMMAND	analog WM_COMMAND
WM_HELP	analog WM_COMMAND
WM_TIMER	Timertick-Notification
WM_CONTROL	schickt Control an Parent/Owner
WM_ERROR	bei "fatalen" Fehlern gesendet
WM_MINMAX	vor Vergrößerungen/Verkleinerungen und Ikonisierungen
Mausnachrichten	
WM_MOUSEMOVE	bei der Bewegung der Maus
WM_BUTTON1DOWN	beim Betätigen des linken Mausknopfes
WM_BUTTON1UP	beim Loslassen des linken Mausknopfes
WM_BUTTON1DBLCLK	Doppel-Click des linken Mausknopfes
WM_BUTTON2DOWN	beim Betätigen des rechten Mausknopfes
WM_BUTTON2UP	beim Loslassen des rechten Mausknopfes
WM_BUTTON2DBLCLK	Doppel-Click des rechten Mausknopfes
WM_BUTTON3DOWN	beim Betätigen des mittleren Mausknopfes
WM_BUTTON3UP	beim Loslassen des mittleren Mausknopfes
WM_BUTTON3DBLCLK	Doppel-Click des mittleren Mausknopfes
Titlebar-Nachrichten	
WM_QUERYTRACKINFO	vor der Bewegung des "Tracking"-Rechtecks
TBM_TRACKMOVE	während der Bewegung des "Tracking"-Rechtecks
TBM_SETSTATE	?
TBM_QUERYSTATE	?
Menu-Nachrichten	
WM_INITMENU	vor der Darstellung eines Menus gesendet
WM_MENUSELECT	nach der Auswahl eines Menus gesendet
WM_MENUEND	nach der Darstellung eines Menus gesendet
MM_INSERTITEM	Menueintrag hinzufügen
MM_REMOVEITEM	Menueintrag entfernen
MM_SELECTITEM	Menueintrag anwählen
MM_QUERYSELITEMID	logische Nummer (ID) des Menueintrags erfragen
MM_QUERYITEM	Menueintrag aus ID erfragen
MM_QUERYITEMTEXT	Text eines Menueintrags aus ID erfragen
MM_SETITEMTEXT	Text eines Menueintrags mit ID setzen
MM_ITEMPOSITIONFROMID	Position eines Menueintrags aus ID
MM_ITEMIDFROMPOSITION	ID eines Menueintrags aus Position
MM_QUERYITEMCOUNT	Anzahl der Menueinträge erfragen
MM_SETITEMATTR	Attribut (Farbe & Zustand) eines Menueintrags setzen
MM_QUERYITEMATTR	Attribut (Farbe & Zustand) eines Menueintrags erfragen
MM_ISITEMVALID	Test, ob gültiger Menueintrag
MM_QUERYITEMTEXTLENGTH	Länge eines Menutextes erfragen
Scrollbar-Nachrichten	
WM_HSCROLL	beim Anklicken des horizontalen Scrollbars gesendet
WM_VSCROLL	beim Anklicken des vertikalen Scrollbars gesendet
SBM_SETSCROLLBAR	?
SBM_SETPOS	Position des "Sliders" setzen
SBM_QUERYPOS	Position des "Sliders" erfragen
SBM_QUERYRANGE	Definitionsbereich des "Sliders" erfragen

Tabelle 3: *Die unterstützten Nachrichten*

des OS/2-PMs, da jedes Window einen lokalen »Farbraum« besitzen kann, der mit Hilfe eines weiteren Class-Styles CS_OWNCOLOR bei der Registrierung der Windowklasse erzeugt wird. Unterbleibt die Spezifikation CS_OWNCOLOR, benutzt das dieser Klasse zugeordnete Window den globalen Farbraum (default). Korrespondierend zur Deklaration eines eigenen Farbraumes je Window, erhalten die Windows bzw. Controls zu bestimmten Zeiten die Color-Notification-Message WM_COLOR.

Class-Styles	Bedeutung Tab.4
CS_SIZEREDRAW	bewirkt die Aktualisierung nach einer Größenveränderung (default)
CS_MOVENOTIFY	Notification-Message bei Veränderung der Windowposition (default)
CS_CLIPSIBLINGS	Berücksichtigung anderer Child-Windows (default)
CS_CLIPCHILDREN	Berücksichtigung der eigenen Child-Windows (default)
CS_PARENTCLIP	(default)
CS_SYNCPAINT	bewirkt eine unmittelbare Aktualisierung
CS_FLASHFRAME	bewirkt ein "Frame-Flashing" bei der Aktivierung (nur QS-PM)
CS_OWNCOLOR	Erzeugung eines eigenen Farbraumes (nur QS-PM)
CS_MENUSEPARATOR	Darstellung einer Trennlinie zwischen Menu- und Client-Window (nur QS-PM)
CS_VIRTSHEET	Erzeugung eines virtuellen Sheets (nur QS-PM)

Tabelle 4: *Die unterstützten Class-Styles*

Beispielsweise erhält jedes Window nach der WM_CREATE-Nachricht eine WM_COLOR-Nachricht, die, falls sie ausgewertet wird, zur Farbkonfiguration des betreffenden Windows oder der gesamten Klasse herangezogen werden kann. In ähnlicher Weise erhalten die Parent- bzw. Owner-Windows eines Menüs oder einer Message-Box vor der Darstellung derselben eine WM_COLOR-Nachricht, wobei der Wert von mp1 darüber entscheidet, ob ein Window, ein Menü oder eine Message-Box der Absender der Nachricht ist (siehe Programm DEMO.C in *Listing 1*).

Class-Styles

Die vom OS/2-PM benötigten Class-Styles sind zum größten Teil implementiert worden, wie man der *Tabelle 4* entnehmen kann. Die Class-Styles CS_SIZEREDRAW, CS_MOVENOTIFY, CS_CLIPSIBLINGS, CS_CLIPCHILDREN und CS_PARENTCLIP sind immer aktiv und zur Zeit nicht zu unterbinden. Die standardmäßige Einstellung der Styles

```
CS_CLIPSIBLINGS | CS_CLIPCHILDREN |
CS_CLIPPARENT
```

hat den Vorteil, daß die Integrität der einzelnen Windows in jeder Lage gesichert ist und es nicht zum Überschreiben anderer Windows durch eine falsch gewählte Clipping-Logik seitens des Entwicklers kommen kann. Dies gilt insbesondere bei der Definition von mehreren Child-Windows und ihren Abkömmlingen. Die Clippingflächen sind im QS-PM der Geometrie des Textbildschirms angepaßt, das heißt alle Clipping-und Update-Bereiche sind rechteckig. Ähnlich verhält es sich bei den Styles

```
CS_SIZEREDRAW | CS_MOVENOTIFY
```

die bewirken, daß automatisch nach jeder Änderung der Größe eines Windows vom PM ein Aktualisierungsbefehl an die davon betroffenen Windows der Applikation geschickt wird (CS_SIZEREDRAW). Der Style CS_MOVENOTIFY bewirkt, daß nach einer Veränderung der Lage eines Windows, diesem immer eine WM_MOVE-Nachricht gesandt wird.

Für alle, die es wissen müssen: Neue Computerfachbücher von iwt.

AutoSketch
Zeichnen, Entwerfen und Illustrieren mit AutoSketch.
Von der einfachen Skizze bis zum anspruchsvollen Dokument

Vom einfachen Skizzieren über das Anlegen und Verwalten von Symbolbibliotheken bis hin zu textlich und graphisch aufbereiteten Dokumentationen werden in diesem Buch alle Anwendungsmöglichkeiten von AutoSketch beschrieben. Der stufenweise Aufbau und die verständliche Darstellung machen Vorkenntnisse nicht erforderlich.

1988. 264 Seiten.
Geb. DM 48,–/Fr. 48.–/
S 374,–
ISBN 3-88322-229-1

IWT Kompakt: MS-DOS
Die Befehlsbibliothek aller MS-DOS-Kommandos bis Version 3.3

Das Buch ist als alphabetisch geordnete Befehlsbibliothek aller internen und externen MS-DOS-Kommandos konzipiert. Jedes Kommando wird detailliert behandelt und neben der Erläuterung von Funktion und Syntax mit Beispielen dargestellt. Für Einsteiger und professionelle MS-DOS-Anwender.

1988. 248 Seiten.
Kart. DM 38,–/Fr. 38.–/
S 296,–
ISBN 3-88322-213-5

WordPerfect
Beispiele-Übungen-Befehlsübersichten

Eine Übersicht über die vielfältigen Möglichkeiten von WordPerfect. Die Grundlagen (z. B. Erfassen, Kopieren, Formatieren) werden ebenso ausführlich beschrieben wie die Sonderfunktionen (z. B. Rechenmöglichkeiten, Rechtschreibprüfung, automatische Indexerstellung). Durch die Unterteilung aller Kapitel in einen Übungs- und Nachschlageteil für Einsteiger und fortgeschrittene Benutzer von WordPerfect eine wertvolle Hilfe.

1988. 280 Seiten.
Geb. DM 58,–/Fr. 58.–/
S 452,–
ISBN 3-88322-206-2

Fortran für PCs und Großrechner f77 – Fortran 77 – MS-Fortran
Für alle Rechner mit dem Betriebssystem MSDOS, CDOS oder UNIX.

Inhalt des Buches ist die komplette Sprachbeschreibung von Fortran 77 sowie ein Sprachvergleich der Betriebssysteme MS-DOS, C-DOS und UNIX. Der 2. Teil geht dann auf alle notwendigen Compiler wie auch auf die Zusatzprogramme ratfor, efl, fsplit, etc. ein.

1988. 616 Seiten.
Geb. DM 78,–/Fr. 78.–/
S 608,–
ISBN 3-88322-180-5

MSDOS im Detail 3
Kurzbeschreibung aller Kommandos, Dienstprogramme und Komponenten von MSDOS 3.30 und 3.31

Beschreibung aller externen/internen Kommandos von MSDOS 3.30, der wichtigen Dienstprogramme (Debug) sowie aller Interrupts und Systemfunktionen. Ein besonderes Kapitel bietet alle wichtigen Informationen zu MSDOS 3.31 (Compaq Deskpro 386-Version). Ein Standardwerk für Software-Anwender und Systemspezialisten.

1988. 584 Seiten.
Geb. DM 78,–/Fr. 78.–/
S 608,–
ISBN 3-88322-203-8

UNIX-Werkzeuge unter MS-DOS
Die UNIX-artige Benutzeroberfläche unter MS-DOS und PC-DOS-Systemen

Dieses Buch gewährt Einblick in die Gemeinsamkeiten von UNIX und MS-DOS in bezug auf Einplatzsysteme. Es zeigt die Entwicklung wesentlicher Unixfunktionen (ls, man, mv, rm, tail, usw.) und deren Integration zu einer UNIX-artigen Kommandooberfläche unter MS-DOS. Nützliche Unix-Werkzeuge stehen dem PC-Anwender somit zur Verfügung.

1988. 367 Seiten.
Geb. DM 58,–/Fr. 58.–/
S 452,–
ISBN 3-88322-219-4

Buch und IWT-SOFTPAC UNIX-Utilities für MS-DOS:
DM 156,–/Fr. 156.–/
S 1.217,–*
Best.-Nr. 92231101

Netzwerkprogrammierung in dBase III plus, Clipper und COBOL.
Planung, Organisation, Programmierung.

Dem Buch liegt ein praktisches Projekt zugrunde, das in dBase III plus und in Clipper entwickelt wurde. Es befaßt sich mit der Planung und Organisation des Netzwerkes und der zu erstellenden Software, mit der Programmierung und dem Testen des Netzwerkes und der Software.

1988. 296 Seiten.
Geb. DM 68,–/Fr. 68.–/
S 530,–
ISBN 3-88322-184-8

Dazu Diskette lieferbar:
DM 98,–/Fr. 98.–/
S 764,–*
Best.-Nr. 92431101

*unverbindliche Preisempfehlung

Das Thema Computer braucht ein tägliches Update. Denn neue Betriebssysteme, neue Software-Versionen, neue Anwendungsmöglichkeiten, neue Programmiertechniken, neue Aspekte, Themen und Tendenzen sind in der Computerbranche an der Tagesordnung.

Wer damit zu tun hat, muß deshalb immer up-to-date sein. Mit iwt-Computer-Fachbücher ist das kein Problem: Unsere Bücher kommen aus der Praxis und unsere Autoren sind vom Fach: Praktiker mit Berufserfahrung, Fachwissen und Sachverstand.

Das garantiert zwei wichtige Dinge: Die fundierte Information zu einem aktuellen Thema. Sieben von über 20 neuen Titeln sehen Sie hier. Jetzt heißt es „Hands on": Fordern Sie unseren Neuheitenprospekt an. Oder holen Sie ihn sich in Ihrer Buchhandlung.



Computer-Fachbücher, die weiterhelfen.

IWT Verlag GmbH, Vaterstetten · Der Fachverlag für Information, Wissenschaft, Technologie
Wendelsteinstraße 3, 8011 Vaterstetten, Telefon (08106) 31017, Telex 5213989 iwt

AUSLIEFERUNG SCHWEIZ: THALI AG, Buchhandlung und Verlag, CH-6285 Hitzkirch, Telefon (041) 85 28 28
AUSLIEFERUNG ÖSTERREICH: ERB-VERLAG Ges. m.b.H. + Co.KG., Amerlingstraße 1, A-1061 Wien 6, Tel. (0222) 587 05 26, Telex 136145

Window- & Frame-Styles	Bedeutung
WS_CLIPCHILDREN	default
WS_CLIPSIBLINGS	default
WS_PARENTCLIP	default
WS_ICON	erzeugt ein Window als Sinnbild (Icon)
WS_VISIBLE	automatische Darstellung nach der Erzeugung
WS_SYNCPAINT	sofortige Aktualisierung
WS_MINIMIZED	wird im "minimized" Zustand kreiert
WS_MAXIMIZED	wird im "maximized" Zustand kreiert
FCF_TITLEBAR	Window mit einfacher Umrandung
FCF_VERTSCROLL	Erzeugung eines vertikalen Scrollbar-Windows
FCF_HORZSCROLL	Erzeugung eines horizontalen Scrollbar-Windows
FCF_SYSMENU	Darstellung des Systemmenus
FCF_SIZEBORDER	Vergrößerung- & Verkleinerungsfunktion
FCF_MINMAX	Vergrößerung-, Verkleinerungs- & Iconfunktion
FCF_MENU	Darstellung eines Applikationsmenus
FCF_STANDARD FCF_TITLEBAR¦FCF_SYSMENU¦FCF_MENU¦FCF_SIZEBORDER¦FCF_MINMAX	
FS_ENCLOSED	Window ist von einem Rahmen umschlossen (QS-PM)
FS_DLGBORDER	Window mit doppelter Umrandung
FS_BORDER	Window mit einfacher Umrandung
FS_ACCELTABLE	Benutzung einer Accelerator-Tabelle
FS_STANDARD	FS_BORDER¦FS_ENCLOSED

Tabelle 5: *Die unterstützten Window- & Frame-Styles und Frame-Flags*

Bei den Styles CS_FLASHFRAME, CS_OWNCOLOR, CS_MENUSEPARATOR und CS_VIRTSHEET handelt es sich um Styles die im OS/2-PM nicht zur Verfügung stehen. CS_FLASHFRAME bewirkt bei der Aktivierung eines Windows ein kurzes Aufblinken der Umrandung dieses Windows. CS_OWNCOLOR erzeugt den oben schon erwähnten, zum jeweiligen Window lokalen Farbraum und CS_MENUSEPARATOR bewirkt die Darstellung einer horizontalen Trennlinie zwischen dem Client- und dem Actionbar-Window eines Standard- bzw. Frame-Windows.

Der Style CS_VIRTSHEET hingegen ist ein Tribut an die virtuellen Features der bislang am PC-Markt erhältlichen Window-Manager und erleichtert die Anpassung damit entwickelter Applikationen an den QS-PM. CS_VIRTSHEET bewirkt die Erzeugung eines Speicherbereiches mit n Spalten und m Zeilen, auf den mit GpiCharStringAt() zugegriffen wird. Der Unterschied besteht darin, daß der mit GpiCharStringAt() ausgegebene Text auf dem virtuellen Bereich gespeichert ist und das am Ende einer Zeile einen Zeilenvorschub und ein Auto-Scroll beim Überschreiten der letzten Zeile stattfindet. Zudem wird die WM_PAINT-Nachricht bei Windows mit Style CS_VIRTSHEET von der Funktion WinDefWindowProc() selbstständig verarbeitet, so daß das Schreiben einer eigenen Paint-Funktion entfällt. Die Ausdehnung dieses Sheets in horizontaler und vertikaler Richtung wird vom Rückgabewert der WM_CREATE-Nachricht festgelegt.

Window- und Framestyles

Von den für den Applikationsprogrammierer wichtigen Window- (WS_*) und Frame-Styles (FS_*) und Frame-Flags (FCF_*) werden die in der *Tabelle 5* aufgelisteten im Sinne des OS/2-PMs unterstützt.

Eine Besonderheit stellt der Frame-Style FS_ENCLOSED dar. Je nach Wahl des Videomodus hat eine Applikation im Textmodus zwischen 80 bis 132 Spalten und 25 bis 60 Zeilen zur Verfügung, eine, verglichen mit der grafischen Variante, deutlich geringere Auflösung. Da in den meisten Fällen das Client-Window des Frame-Windows zur Ausgabe von Information benutzt wird, reduziert sich die maximale auf dem Bildschirm sichtbare Höhe im ungünstigsten Fall um weitere fünf Zeilen für Umrandung (unten & oben), Titlebar-, Actionbar- (oben) und Scrollbar-Window (unten). Durch die Benutzung des Frame-Styles

```
FS_STANDARD & ~FS_ENCLOSED
```

entfällt die explizite Umrandung des Frame-Windows, da Titlebar- und Scrollbar-Windows auf der Umrandung selbst dargestellt werden und so zwei Zeilen und Spalten gewonnen werden.

Von den Controls des Frame-Windows stehen dem DOS-Programmierer die Titlebar-, die vertikalen und horizontalen Scrollbar-, die Action-Menu-, die System-Menu-, die Sizebox-, Sizeborder- und die Minmax-Control zur Verfügung, deren Funktionsumfang zu dem vom OS/2-PM identisch ist.

Die Tastatur-, Timer- und Mausschnittstelle

Neben der Aufrufkonvention der Bildschirmfunktionen geben häufig auch die Tastatur- und Mausschnittstellen Anlaß zur mangelnder Portabilität. Nicht so beim QS-PM, der sich hierbei praktisch vollständig kompatibel zum OS/2-PM verhält. Die Tastaturschnittstelle der Applikation wird durch die WM_CHAR-Nachricht hergestellt, wobei nur die Bits KC_SCANCODE, KC_LONEKEY, KC_DEADKEY, KC_COMPOSITE, KC_INVALIDCOMP zur Zeit nicht unterstützt werden. Die Verwaltung der Umlaute, die normalerweise über die Bits KC_DEADKEY und KC_COMPOSITE dekodiert werden, wird im QS-PM zur Zeit mit KC_CHAR kodiert, so daß zum Beispiel die Taste »Ä« mit dem Wert 142 als ASCII-Erweiterung kodiert ist. Die Maus- und Timerschnittstelle hingegen ist 100% kompatibel zu der des OS/2-PMs.

Die Speicherverwaltung

Die dynamische Speicherverwaltung ist Teil des QuickStep Resource-Managers, der die meisten der Hardware- und betriebssystemspezifischen Komponenten enthält. Die Speicherverwaltung ist OS/2-kompatibel und erfolgt somit über Handles bzw. Selektoren. Auf diese Weise konnten wir die Unterstützung von virtuellem Speicher (Platte) und EMS-Speicher vor der Applikation verbergen und portabel halten, so daß die Applikationsschnittstelle für MS-Windows, DOS, OS/2 und UNIX immer identisch und die Allokationsart über die Konstanten SEG_* kodiert ist. Speicherblöcke die »discardable« (SEG_DISCARDABLE) sind oder im EMS-Speicherbereich (SEG_EMS) liegen, müssen vor Gebrauch »gelocked« und nach Gebrauch wieder »unlocked« werden.

Der Resource Compiler

Wie sein OS/2-Pendant verfügt auch der QuickStep Presentation Manager über einen Resource-Compiler zur Wartung und Definition von Strings, Menüs und Accelerators (Dialog-Ressourcen sind zur Zeit nicht implementiert). Im Unterschied zum OS/2-PM erzeugt der QS-PM jedoch wahlweise C-, Modula-2- oder Pascal-Code, der nachträglich noch kompiliert werden muß oder eine Datei, die in kompakter Form die Ressourcedefinitionen enthält und von der Applikation zur Laufzeit geladen werden können. Somit entspricht der C-Code-Erzeugung die Option PRELOAD und der Dateivarianten die Option LOAD ON CALL des OS/2-PM. Diese Unterscheidung war notwendig, da wir das DOS-EXE-Format nicht modifizieren wollten. Das *Listing 2* zeigt einen Ausschnitt der zum Beispielprogramm gehörenden Ressourcedatei DEMO.RC und *Listing 3* die vom Ressource-Compiler erzeugte Datei DEMO.RCC. Interessant ist im Zusammenhang mit den Menüdefinitionen, daß der QS-PM auch beliebig geschachtelte Menüdefinitionen verarbeiten kann.

Resümee

Zusammenfassend kann gesagt werden, daß der QuickStep-Presentation Manager zur Entwicklung von Applikationen unter DOS zum Zwecke der späteren leichten Portierung nach OS/2 und UNIX entwickelt wurde. Da in der DOS-Version keinerlei grafische Unterstützung zur Verfügung steht, dürften die meisten damit geschriebenen Applikationen mit dem OS/2-Presentation Manager unverändert laufen. Der umgekehrte Weg eine vollständige OS/2-PM Applikation mit ausgeprägter Grafikschnittstelle unter DOS mit dem QS-PM laufen zu lassen, dürfte einige Modifikationen am Programm erfordern. Die bislang fehlende Unterstützung von Dialogboxen und Clipboardfunktionen soll bis zum Herbst 1989 nachgeholt sein. In dieser Version dürfte der QS-PM eine echtes Argument zur Entwicklung von den Microsoft Presentation Manager unterstützenden Anwendungen sein, da dann für die wichtigsten Betriebssysteme DOS, OS/2 und UNIX sowohl eine Grafikschnittstelle (OS/2 und UNIX), wie auch eine Textschnittstelle (DOS und UNIX) zur Verfügung steht und somit Portabilitätshemmnisse nicht mehr vorhanden sind.

Rainer Wallwitz

Der Autor ist Geschäftsführer der Fa. Lauer & Wallwitz GmbH und dort für die Entwicklung von MS-Windows-, OS/2- und Presentation-Manager-Tools und Applikationen verantwortlich. Das beschriebene Produkt QuickStep Presentation Manager kann auf der CeBIT 89 am Stand von Lauer & Wallwitz (Halle 6, Stand H20, direkt gegenüber von Microsoft) besichtigt werden.

```
/*****************************************************************************
 *       DEMO.C: Demonstrationsprogramm der QuickStep Presentation-Manager
 *       Bibliothek für DOS
 *       (C) Copyright LAUER & WALLWITZ,
 *       6200 Wiesbaden, Kirchgasse 24, Tel. 06121 / 30 10 65
 *****************************************************************************/

#include <qrmodel.h>
#include <string.h>
#include "appli.h"

MRESULT EXPENTRY ClientWndProc( HWND, USHORT, MPARAM, MPARAM );
MRESULT EXPENTRY ClientChildWndProc( HWND, USHORT, MPARAM, MPARAM );
MRESULT EXPENTRY WndProc( HWND, USHORT, MPARAM, MPARAM );

static void RegisterClasses( void );
static void ClientPaint( HPS, USHORT, USHORT );
static void ChildPaint( HPS );

static PSZ Sheet[] = {
 "══════════════════════════════════════════════════════════════════════",
    .
    .
    .
 "══════════════════════════════════════════════════════════════════════"
};
static usSheetLines = 31;

static PSZ  pszClientClassName      = "CLIENT";
static PSZ  pszClientChildClassName = "CHILD";
static PSZ  pszFrame2ClassName      = "FRAME2";
static char QuitBuffer[50];
static char TermBoxBuffer[50];
static HAB  hAB;

/*** main *****************************************************/

int CDECL main( argc, argv )
 int  argc;
 PSZ argv[];
  {
   HMQ    hMQ;
   QMSG   Msg;
   USHORT usChoice;
   BOOL   fContinue;
   HWND   hFrame,hClient;
   ULONG  flFrameFlags = FCF_MINMAX  | FCF_VERTSCROLL | FCF_HORZSCROLL |
                         FCF_SYSMENU | FCF_MENU;

                              /*  Hintergrundzeichen und -attribut setzen  */
   WinCustomizeQS( CUST_SET,             /*  Nicht OS/2-PM kompatibel      */
                   CUST_VIO_BACKGROUND,
                   MAKEULONG(0x33B1,0) );
   WinCustomizeQS( CUST_SET,      /*  Aktivierung des 80x43 Zeilen-Modus (EGA) */
                   CUST_VIO_MODE,        /*  Nicht OS/2-PM kompatibel      */
                   VIO_EGA | VIO_COLOR /*| VIO_80_43*/ );
   hAB = WinInitialize();                /*  PM initialisieren      */
   hMQ = WinCreateMsgQueue( hAB, 0 );  /*  System-Queue erzeugen  */

      /*  Ressource-String IDS_WANT_QUIT laden  */
   WinLoadString( hAB, NULL, IDS_WANT_QUIT, 50, QuitBuffer ),

      /*  Ressource-String IDS_TERMBOX laden  */
   WinLoadString( hAB, NULL, IDS_TERMBOX,   40, TermBoxBuffer ),

      /*  Window-Klassen registrieren  */
   RegisterClasses();

      /*  Frame-Window kreieren  */
   hFrame = WinCreateStdWindow( HWND_DESKTOP,
                         WS_VISIBLE | FS_BORDER  | FS_DLGBORDER  | FS_SIZEBORDER |
                         FS_ENCLOSED,   /*  nicht OS/2-PM kompatibel, zusätzlich */
                         &flFrameFlags,
                         pszClientClassName,
                         "QUICKSTEP - TOOLS",
                         0L,
                         NULL,
                         IDM_APPLMENU3,
                         &hClient );

         /*  Message-Loop  */
   do {
     if ( !(fContinue=WinGetMsg(hAB,&Msg,NULL,0,0)) ) {

/*  Messagebox aktivieren und fragen, ob Programm beendet werden soll  */
       usChoice=WinMessageBox( HWND_DESKTOP , HWND_DESKTOP,
                               QuitBuffer,
                               TermBoxBuffer,
                               0,
                               MB_OKCANCEL | MB_DEFBUTTON2 );
      fContinue = (usChoice == MBID_OK ) ? FALSE : TRUE;
     }

    else WinDispatchMsg( hAB, &Msg );
   } while( fContinue );

     /*  Frame-Window zerstören  */
   WinDestroyWindow( hFrame );
     /*  System-Queue zerstören  */
   WinDestroyMsgQueue( hMQ );
     /*  PM verlassen */
   WinTerminate( hAB );
   return 0;
  }
```

Listing 1: `DEMO.C`

```
/*** ClientWndProc ****************************************************/

MRESULT EXPENTRY ClientWndProc( hWnd, message, mpParam1 , mpParam2 )
 HWND   hWnd;
 USHORT message;
 MPARAM mpParam1,mpParam2;
  {
   HPS    hPS;
   RECT   rc;
   ULONG  flFrameFlags;
   static USHORT usLow,usHigh;
   static HWND   hFrame,hClient;

   switch( message )
    {
    case WM_COLOR : {   /*  Farb- und Stylekonfiguration, nur QS-PM  */
          PCOLORSTRUCT pColor = (PCOLORSTRUCT)mpParam2;
          if ( LOUSHORT(mpParam1) == COLOR_WND ) {
             /*  Installation des Windows  */
             pColor->yActiveBorder        = 30;
             pColor->yInactiveBorder      = 7;
             pColor->wActiveCaption       = 0x1820;
             pColor->wInactiveCaption     = 0x1820;
             pColor->yActiveCaptionText   = 112;
             pColor->yInactiveCaptionText = 7;
             pColor->usBackground         = 0x0720;
             pColor->bSysCtlAttr          = 112;
             pColor->bMinMaxCtlAttr       = 112;
             pColor->CTL.pScrl->yVThumbAttr = 0x1f;
             pColor->CTL.pScrl->yHThumbAttr = 0x1f;
             pColor->CTL.pScrl->yVBackAttr  = 0x17;
             pColor->CTL.pScrl->yHBackAttr  = 0x17;
             pColor->CTL.pScrl->yScrlVBoundAttr = 0x1f;
             pColor->CTL.pScrl->yScrlHBoundAttr = 0x1f;
            }
          else if ( LOUSHORT(mpParam1) == COLOR_MBOX ) {
               /*  Installation einer Messagebox */
               pColor->CTL.pMBCol->bHeaderAttr      = 1f;
               pColor->CTL.pMBCol->bFrameAttr       = 7;
               pColor->CTL.pMBCol->bActButFrmAttr   = 7;
               pColor->CTL.pMBCol->bInactButFrmAttr = 7;
               pColor->CTL.pMBCol->bActButTextAttr  = 7;
               pColor->CTL.pMBCol->bInactButTextAttr= 7;
              }
         }
         break;

    case WM_CREATE :
         WinQueryWindowRect( hWnd, &rc );
         usLow  = 0;
         usHigh = rc.yBottom-rc.yTop+1;

         /*  Lage und Größe des Frame-Windows, d.h. des Owners umsetzen  */
         WinSetWindowPos( WinQueryWindow(hWnd,QW_OWNER,FALSE),
                          NULL, 5, 5,
                          40, 20,
                          SWP_SIZE | SWP_MOVE );

         /*  eine weiteres Desktop Window erzeugen  */
         flFrameFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_MENU;
         hFrame = WinCreateStdWindow( HWND_DESKTOP,
                            FS_SIZEBORDER |
                            FS_ENCLOSED, /* nicht OS/2-PM kompatibel, zusätzlich */
                            &flFrameFlags,
                            pszFrame2ClassName,
                            "Frame2-Window",
                            0L,
                            NULL,
                            IDM_APPLMENU1,
                            &hClient );
         break;

    case WM_DESTROY :   /*  das bei WM_CREATE erzeugte Window zerstören  */
         WinDestroyWindow( hFrame );
         break;

    case WM_SIZE :       /*  Höhe des Client-Windows ermitteln  */
         usHigh = usLow+HIUSHORT((ULONG)mpParam1);
         break;

    case WM_VSCROLL :   /*  Scroll-Notification auswerten und  */
                        /*  die Client-Area aktualisieren      */

         switch( HIUSHORT(mpParam2) )
          {
           case SB_LINEUP     :
           case SB_LINELEFT :
                if ( usLow > 0 ) {
                   usLow--;
                   usHigh--;
                   WinSendMsg(hWnd,WM_PAINT,0L,0L);
                  }
                break;
           case SB_LINEDOWN    :
           case SB_LINERIGHT :
                if ( usHigh < usSheetLines-1 ) {
                   usLow++;
                   usHigh++;
                   WinSendMsg(hWnd,WM_PAINT,0L,0L);
                  }
                break;
          }
         break;

    case WM_ACTIVATE :
         if ( HIUSHORT((ULONG)mpParam1)==FALSE ) return 0L;
                  /*  Client-Window erhält den Input-Focus    */
         WinSetFocus( HWND_DESKTOP, hWnd );
         break;
```

Listing 1: *(Fortsetzung)*

```
    case WM_COMMAND :
         switch( LOUSHORT((ULONG)mpParam1) )
          {
           case IDM_QUIT :      /*  Terminierung des Programmes  */
                WinPostMsg( hWnd, WM_QUIT, 0L, 0L );
                break;

           case IDM_OPENWND :  /*  Öffnen des Frame-Windows     */
                WinShowWindow( hFrame, TRUE );
                WinUpdateWindow( hFrame );
                break;

           case IDM_CLOSEWND : /*  Schließen des Frame-Windows  */
                WinShowWindow( hFrame, FALSE );
                break;
          }
         break;

    case WM_PAINT :             /*  Client-Paint Funktion        */
         hPS = WinBeginPaint( hWnd, NULL, NULL );
         ClientPaint( hPS, usLow, usHigh );
         WinEndPaint( hPS );
         break;

    default : return WinDefWindowProc( hWnd, message, mpParam1, mpParam2 );
   }

  return 0L;
 }

/*** ClientChildWndProc ***********************************************/

MRESULT EXPENTRY ClientChildWndProc( hWnd, message, mpParam1 , mpParam2 )
 HWND hWnd;
 USHORT message;
 MPARAM mpParam1,mpParam2;
  {
   HPS hPS;
   switch( message )
   {

    case WM_CREATE :

                  /* aktuelle Farbe auf Attribut 112 setzen  */
                  /*  nicht OS/2-PM kompatibel               */

         WinSetTextColorQS( WinGetPS(hWnd), 112 );
         break;

    case WM_COLOR :   /*  Farb- und Stylekonfiguration, nur QS-PM  */

         if ( LOUSHORT((ULONG)mpParam1) == COLOR_WND ) {
           PCOLORSTRUCT pColor = (PCOLORSTRUCT)mpParam2;
           pColor->yActiveBorder    = 0x0f;
           pColor->wActiveCaption   = 0x1820;
           pColor->wInactiveCaption = 0x1820;
           pColor->bMinMaxCtlAttr   = 112;
          }
         break;

    case WM_PAINT :
         hPS = WinBeginPaint( hWnd, NULL, NULL );

            /*  Window-Rechteck mit WinFillRect mit 0x7020 fuellen   */
            /*  Wert des letzten Parameters nicht OS/2-PM kompatibel */

         WinFillRect( hPS, NULL, 0x7020 );
         ChildPaint( hPS );
         WinEndPaint( hPS );
         break;

    default : return WinDefWindowProc( hWnd, message, mpParam1, mpParam2 );
   }

  return 0L;
 }

/*** WndProc **********************************************************/

MRESULT EXPENTRY WndProc( hWnd, message, mpParam1 , mpParam2 )
 HWND hWnd;
 USHORT message;
 MPARAM mpParam1,mpParam2;
  {
   HPS    hPS;
   ULONG  flFrameFlags;
   static HWND hFrame,hClient;
   switch( message )
   {
    case WM_CREATE :
                  /*  aktuelle Zeichenfarbe setzen  */
                  /*  nicht OS/2-PM kompatibel      */
         WinSetTextColorQS( WinGetPS(hWnd), 0x20 );
                  /*   Frame-Window al Child-Window des Clients erzeugen  */
         flFrameFlags = FCF_MINMAX  | FCF_VERTSCROLL | FCF_HORZSCROLL |
                        FCF_SYSMENU | FCF_MENU;
         hFrame = WinCreateStdWindow( hWnd,
                            FS_DLGBORDER | FS_SIZEBORDER |
                            FS_ENCLOSED, /* nicht OS/2-PM kompatibel, zusätzlich */
                            &flFrameFlags,
                            pszClientChildClassName,
                            "Child-Window",
                            0L,
                            NULL,
                            IDM_APPLMENU2,
                            &hClient );
         break;
```

Listing 1: *(Fortsetzung)*

```
    case WM_DESTROY :
         WinDestroyWindow( hFrame );
         break;


    case WM_COMMAND :
         switch( LOUSHORT((ULONG)mpParam1) )
         {

          case IDM_OPENCHILD:   /*  Öffnen des Frame-Windows     */
               WinShowWindow( hFrame, TRUE );
               WinUpdateWindow( hFrame );
               break;

          case IDM_CLOSECHILD:  /*  Schließen des Frame-Windows  */
               WinShowWindow( hFrame, FALSE );
               break;
         }
         break;


    case WM_COLOR :  /*  Farb- und Stylekonfiguration, nur QS-PM  */
         if ( LOUSHORT((ULONG)mpParam1) == COLOR_WND ) {
           PCOLORSTRUCT pColor = (PCOLORSTRUCT)mpParam2;
           pColor->wActiveCaption   = 0x4020;
           pColor->wInactiveCaption = 0x1820;
         }
         break;


    case WM_PAINT :
         hPS = WinBeginPaint( hWnd, NULL, NULL );

            /*  Window-Rechteck mit WinFillRect mit 0x2020 fuellen   */
            /*  Wert des letzten Parameters nicht OS/2-PM kompatibel */

         WinFillRect( hPS, NULL, 0x2020 );
         ChildPaint( hPS );
         WinEndPaint( hPS );
         break;

    default :
         return WinDefWindowProc( hWnd, message, mpParam1, mpParam2 );
   }

  return 0L;
 }


/*** ClientPaint ****************************************************/

static void ClientPaint( hPS, usL, usH )
 HPS hPS;
 USHORT usL,usH;
  {
   USHORT usNdx = usL;
   POINTL Pt;

   usH  = (usH<=usSheetLines) ? usH : usSheetLines;
   Pt.x = Pt.y = 0;
   for(usNdx=usL; usNdx<usH; usNdx++) {
     GpiCharStringAt( hPS, &Pt, 70L, Sheet[usNdx] );
     Pt.y++;
   }
  }



/*** ChildPaint *****************************************************/

static void ChildPaint( hPS )
 HPS hPS;
  {
   USHORT usNdx;
   POINTL Pt:

   Pt.x = 0;
   for(usNdx=12; usNdx<usSheetLines; usNdx++) {
     Pt.y = usNdx-12;
     GpiCharStringAt( hPS, &Pt, 70L, Sheet[usNdx] );
   }
  }



/*** RegisterClasses ************************************************/

static void RegisterClasses()
  {
   WinRegisterClass( hAB,
                     pszClientClassName,
                     ClientWndProc,
                     CS_DEFAULT | CS_OWNCOLOR,  /*  Klasse mit eigenem Farbraum  */
                     0 );


   WinRegisterClass( hAB,
                     pszClientChildClassName,
                     ClientChildWndProc,
                     CS_DEFAULT | CS_OWNCOLOR,  /*  Klasse mit eigenem Farbraum  */
                     0 );


   WinRegisterClass( hAB,
                     pszFrame2ClassName,
                     WndProc,                   /*  Klasse ohne eigenen Farbraum */
                     CS_DEFAULT,
                     0 );

  }
```

Listing 1: *(Ende)*

```
/***************************************************************************
 *       DEMO.RC
 *       Ressourcedatei des Programms DEMO.C
 ***************************************************************************/


MODULE InstallResourcesQS
 BEGIN


#include <qspm.h>
#include "appli.h"


CODE #include <appli.h> CODE END


STRINGTABLE DemoStrings
   {
    IDS_WANT_QUIT  ,"\nDo you want to terminate this demonstration ?\n"
    IDS_TERMBOX    , "Termination Box"
   }


MENU "Applikationsmenu1", ApplMenuItems, IDM_APPLMENU1
   {
    STYLE HORIZONTAL
    MENUITEM "~Öffne Child"    ,   IDM_OPENCHILD
    MENUITEM "~Schliesse Child",   IDM_CLOSECHILD
   }


MENU "Applikationsmenu 2", ApplMenuItems2, IDM_APPLMENU2
   {
    STYLE HORIZONTAL
    MENUITEM "~Create" , IDM_CREATE
    MENUITEM "~Modify" , IDM_MODIFY
    MENUITEM "~Delete" , IDM_DELETE
    MENUITEM "~Switch" , IDM_SWITCH, MIA_DISABLED
    MENUITEM "~Append" , IDM_APPEND, MIA_DISABLED
   }


MENU "Applikationsmenü 3", ApplMenuItems3, IDM_APPLMENU3
   {
    STYLE  HORIZONTAL
    MENUITEM "~Öffnen",          IDM_OPENWND
    MENUITEM "~Schließen",       IDM_CLOSEWND

    SUBMENU "AutoMenu", AutoMenu, IDM_AUTOMENU,     IDM_AUTO
     {
     STYLE     VERTICAL
     TITLE TOP "CARS"
     MENUITEM "~BMW",             IDM_BMW
     MENUITEM "~MERCEDES BENZ",   IDM_BENZ
     MENUITEM "~OPEL",            IDM_OPEL
     MENUITEM "~AUDI",            IDM_AUDI
     MENUITEM "~PORSCHE",         IDM_PORSCHE
     MENUITEM "~FORD",            IDM_FORD
     }

    SUBMENU "ComputerMenu", ComputerMenu, IDM_COMPUTERMENU, IDM_COMPUTER
     {
     STYLE       VERTICAL
     TITLE TOP "Computers"
     MENUITEM "~COMPAQ",    IDM_COMPAQ
     MENUITEM "~IBM",       IDM_IBM

     SUBMENU "BankenMenu", BankenMenu,  IDM_BANKENMENU, IDM_FUJITSU
       {
       STYLE      VERTICAL
       TITLE TOP "German Banks"
       MENUITEM "~Commerzbank",   IDM_CBANK
       MENUITEM "~Deutsche Bank", IDM_DTBANK

       SUBMENU  "StadtMenu", StadtMenu, IDM_STADTMENU,  IDM_DRBANK
         {
         STYLE     VERTICAL
         TITLE TOP "Cities"
         MENUITEM "~Hamburg",   IDM_HAMBURG
         MENUITEM "~Dortmund",  IDM_DORTMUND
         MENUITEM "~Wiesbaden", IDM_WIESBADEN, MIA_CHECKED
         MENUITEM "Main~z",     IDM_MAINZ
         MENUITEM "~Berlin",    IDM_BERLIN
         MENUITEM "~Frankfurt", IDM_FRANKFURT
         MENUITEM SEPARATOR
         MENUITEM "~München",   IDM_MUNCHEN
         MENUITEM "~Hanno~ver", IDM_HANNOVER
         MENUITEM "B~remen",    IDM_BREMEN
         }
       MENUITEM "~Hypo Bank",      IDM_HBANK
       }
     MENUITEM "~DEC",        IDM_DEC
     MENUITEM "~SUN",        IDM_SUN
     MENUITEM "~APOLLO",     IDM_APOLLO
     }
   MENUITEM "FlugzeugMenu", IDM_FLUGZEUG
   MENUITEM "~Quit",        IDM_QUIT
   }


ACCELTABLE AccelTable1, IDM_APPLMENU3
   {
    VK_F1, IDM_OPENWND,  KC_VIRTUALKEY                    /* F2          */
    VK_F2, IDM_APOLLO,   KC_VIRTUALKEY|KC_SHIFT           /* SHIFT F2    */
    113,   IDM_QUIT,     KC_CHAR|KC_ALT                   /* ALT q       */
   }

 END
```

Listing 2: DEMO.RC

```
#include <qrmodel.h>
#include <pmrcmenu.h>
#include <pmrcaccl.h>
#include <appli.h>

static PSZ DemoStrings[] = {
 "\nDo you want to terminate this demonstration ?\n",
 "Termination Box",
  NULL
};

static MENUITEMTEMPLATE ApplMenuItems[] = {
 {"~Öffne Child" , NULLH, MIS_STRING             ,IDM_OPENCHILD  ,NULLH     },
 {"~Schliesse Child", NULLH, MIS_STRING          ,IDM_CLOSECHILD ,NULLH     },
 {NULL           , NULLH, MS_HORIZONTAL          ,MENU_END       ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE ApplMenuItems2[] = {
 {"~Create"      , NULLH, MIS_STRING             ,IDM_CREATE     ,NULLH     },
 {"~Modify"      , NULLH, MIS_STRING             ,IDM_MODIFY     ,NULLH     },
 {"~Delete"      , NULLH, MIS_STRING             ,IDM_DELETE     ,NULLH     },
 {"~Switch"      , NULLH, MIS_STRING|MIA_DISABLED,IDM_SWITCH      ,NULLH    },
 {"~Append"      , NULLH, MIS_STRING|MIA_DISABLED,IDM_APPEND      ,NULLH    },
 {NULL           , NULLH, MS_HORIZONTAL          ,MENU_END       ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE AutoMenu[] = {
 {"~BMW"         , NULLH, MIS_STRING             ,IDM_BMW        ,NULLH     },
 {"~MERCEDES BENZ", NULLH, MIS_STRING            ,IDM_BENZ       ,NULLH     },
 {"~OPEL"        , NULLH, MIS_STRING             ,IDM_OPEL       ,NULLH     },
 {"~AUDI"        , NULLH, MIS_STRING             ,IDM_AUDI       ,NULLH     },
 {"~PORSCHE"     , NULLH, MIS_STRING             ,IDM_PORSCHE    ,NULLH     },
 {"~FORD"        , NULLH, MIS_STRING             ,IDM_FORD       ,NULLH     },
 {NULL           , NULLH, MS_VERTICAL            ,MENU_END       ,NULLH     },
 {"CARS"         , NULLH, MIS_TOPTITLE           ,0              ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE StadtMenu[] = {
 {"~Hamburg"     , NULLH, MIS_STRING             ,IDM_HAMBURG    ,NULLH     },
 {"~Dortmund"    , NULLH, MIS_STRING             ,IDM_DORTMUND   ,NULLH     },
 {"~Wiesbaden"   , NULLH, MIS_STRING|MIA_CHECKED,IDM_WIESBADEN   ,NULLH     },
 {"Main~z"       , NULLH, MIS_STRING             ,IDM_MAINZ      ,NULLH     },
 {"~Berlin"      , NULLH, MIS_STRING             ,IDM_BERLIN     ,NULLH     },
 {"~Frankfurt"   , NULLH, MIS_STRING             ,IDM_FRANKFURT  ,NULLH     },
 {NULL           , NULLH, MIS_SEPARATOR          ,NULLH          ,NULLH     },
 {"~München"     , NULLH, MIS_STRING             ,IDM_MUNCHEN    ,NULLH     },
 {"~Hanno~ver"   , NULLH, MIS_STRING             ,IDM_HANNOVER   ,NULLH     },
 {"B~remen"      , NULLH, MIS_STRING             ,IDM_BREMEN     ,NULLH     },
 {NULL           , NULLH, MS_VERTICAL            ,MENU_END       ,NULLH     },
 {"Cities"       , NULLH, MIS_TOPTITLE           ,0              ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE BankenMenu[] = {
 {"~Commerzbank" , NULLH, MIS_STRING             ,IDM_CBANK      ,NULLH     },
 {"~Deutsche Bank", NULLH, MIS_STRING            ,IDM_DTBANK     ,NULLH     },
 {"StadtMenu"    , NULLH, MIS_SUBMENU            ,IDM_STADTMENU  ,IDM_DRBANK},
 {"~Hypo Bank"   , NULLH, MIS_STRING             ,IDM_HBANK      ,NULLH     },
 {NULL           , NULLH, MS_VERTICAL            ,MENU_END       ,NULLH     },
 {"Cities"       , NULLH, MIS_TOPTITLE           ,0              ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE ComputerMenu[] = {
 {"~COMPAQ"      , NULLH, MIS_STRING             ,IDM_COMPAQ     ,NULLH     },
 {"~IBM"         , NULLH, MIS_STRING             ,IDM_IBM        ,NULLH     },
 {"BankenMenu"   , NULLH, MIS_SUBMENU            ,IDM_BANKENMENU ,IDM_FUJITSU},
 {"~DEC"         , NULLH, MIS_STRING             ,IDM_DEC        ,NULLH     },
 {"~SUN"         , NULLH, MIS_STRING             ,IDM_SUN        ,NULLH     },
 {"~APOLLO"      , NULLH, MIS_STRING             ,IDM_APOLLO     ,NULLH     },
 {NULL           , NULLH, MS_VERTICAL            ,MENU_END       ,NULLH     },
 {"Cities"       , NULLH, MIS_TOPTITLE           ,0              ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static MENUITEMTEMPLATE ApplMenuItems3[] = {
 {"~Öffnen"      , NULLH, MIS_STRING             ,IDM_OPENWND    ,NULLH     },
 {"~Schließen"   , NULLH, MIS_STRING             ,IDM_CLOSEWND   ,NULLH     },
 {"AutoMenu"     , NULLH, MIS_SUBMENU            ,IDM_AUTOMENU   ,IDM_AUTO },
 {"ComputerMenu" , NULLH, MIS_SUBMENU            ,IDM_COMPUTERMENU,IDM_COMPUTER},
 {"FlugzeugMenu" , NULLH, MIS_STRING             ,IDM_FLUGZEUG   ,NULLH     },
 {"~Quit"        , NULLH, MIS_STRING             ,IDM_QUIT       ,NULLH     },
 {NULL           , NULLH, MS_HORIZONTAL          ,MENU_END       ,NULLH     },
 {NULL           , NULLH, 0                      ,MENU_LASTENTRY ,NULLH     },
};

static ACCELITEMSTRUCT AccelTable1[] = {
 { VK_F1  , IDM_OPENWND    , KC_VIRTUALKEY          },
 { VK_F2  , IDM_APOLLO     , KC_VIRTUALKEY|KC_SHIFT },
 { 113    , IDM_QUIT       , KC_CHAR|KC_ALT         },
 { VK_NULL, 0              , 0L                     }
};

void APIENTRY InstallResourcesQS() {
 InsertMenuResourceQS( StadtMenu, IDM_STADTMENU );
 InsertStringTableQS( DemoStrings, IDS_WANT_QUIT );
 InsertMenuResourceQS( ApplMenuItems, IDM_APPLMENU1 );
 InsertMenuResourceQS( BankenMenu, IDM_BANKENMENU );
 InsertMenuResourceQS( AutoMenu, IDM_AUTOMENU );
 InsertMenuResourceQS( ApplMenuItems2, IDM_APPLMENU2 );
 InsertMenuResourceQS( ApplMenuItems3, IDM_APPLMENU3 );
 InsertAccelTableQS( AccelTable1, IDM_APPLMENU3 );
 InsertMenuResourceQS( ComputerMenu, IDM_COMPUTERMENU );
}
```

Listing 3: `DEMO.RRC`

Impressum

Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats.

Herausgeber, verantwortlich und Anschrift der Redaktion:
Microsoft GmbH, Redaktion *Microsoft System Journal*,
Erdinger Landstr. 2, D-8011 Aschheim-Dornach
Telefon: 089 / 46107-0, Teletex/Telex: (17) 89 83 28, Telefax: 90 63 55

Redaktion:
Günter Jürgensmeier, Haar und Hartmut Niemeier, Wildenberg

Mitarbeiter dieser Ausgabe:
Chip Anderson, Marcellus Buchheit, Günter Jürgensmeier, Hartmut Niemeier, Michael Tischer, Rainer Wallwitz, Kevin Welch, David West

Manuskripteinsendungen:
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.

Titelgestaltung: Hermann Menig
Anzeigenverkauf: Marianne Nuß
Druck und Abonnements:
schury praxisformulare GmbH, Postfach 270, D-8200 Rosenheim
Bezugspreise: Das Einzelheft kostet DM 19,80. Der Abonnementpreis beträgt DM 115,- für 6 Ausgaben und DM 210,- für 12 Ausgaben. Zu den einzelnen Ausgaben ist zum Preis von DM 19,80 eine Diskette mit allen Listings erhältlich. Das Abonnement inklusive Diskette kostet DM 230,- für 6 Ausgaben und DM 420,- für 12 Ausgaben. In den Preisen enthalten sind Mehrwertsteuer, Versandkosten und Zustellgebühren. Auslandsbezug auf Anfrage. Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.
Bezugsmöglichkeiten: In Buchhandlungen und im Computer-Fachhandel. Abonnements und Einzelbestellungen: schury GmbH.

Das *Microsoft System Journal* wird mit Microsoft Word 4.0 geschrieben und gestaltet. Der Ausdruck erfolgt mit den HP-Softfonts AD und AF und dem Programm- und Schriftenpaket *DocuJet* auf einem HP-LaserJet Series II.

Alle Perspektiven von CAD.

HANNOVER MESSE CeBIT '89
Welt-Centrum Büro · Information · Telekommunikation
8. – 15. MÄRZ 1989
Halle 7 · Stand E 14

Die „C" Techniken bringen neue Perspektiven für Konstrukteure, Ingenieure, Architekten und Designer. CAD, CAM, CAP oder CAE gehören schon zum Sprachgebrauch in vielen Branchen. Denn immer mehr werden diese Techniken auch auf Computern einsetzbar, die jedermann nutzen kann: Auf Personal Computern und Workstations.

Konstruieren und Entwickeln per Computer, Zeichnen und Planen am Bildschirm sind die zukunftsweisenden Arbeitstechniken. Und es sind genau die Themen des neuen AUTOCAD Magazins: Die „C"-Techniken mit allen ihren Perspektiven.

In Theorie und Praxis. In Anwender- und Erfahrungsberichten. Mit Hard- und Software-Übersichten. Mit Beschreibungen und Tests. Mit Einsatzbeispielen und Einsatzvoraussetzungen.

Mit allen Themen eben, die zu diesem Thema gehören. Im AUTOCAD Magazin wird vor Ihnen das ganze Spektrum dieser Möglichkeiten aufgeblättert. Denn es ist das Magazin, das sich speziell mit den „C"-Techniken auf Personal Computern und Workstations beschäftigt. Und das deshalb ganz speziell und detailliert auf diese Thematik eingehen kann.

Wer von Berufs wegen dieses Thema im Auge haben muß, ist mit dem AUTOCAD-Abonnement automatisch im Bilde. Denn es ist ein Abonnement auf wichtige und aktuelle Informationen.

Eine Zeitschrift aus dem IWT-Verlag.



Bestellcoupon

Hiermit bestelle/n ich/wir:

- ☐ das AUTOCAD-Magazin im Abonnement (vorerst 4 Ausgaben pro Jahr) zum Abopreis von DM 46,– (4 x 9,– zzgl. Versandkosten). Ich kann das Abonnement 8 Wochen vor Ende des Bezugszeitraumes kündigen.
- ☐ ein Einzelheft zum Preis von DM 14,50 (DM 12,– + DM 2,50 Versandkosten)
- ☐ gegen Vorauszahlung auf unser Post-Girokonto München, Kto.-Nr. 413484-807, BLZ 700 100 80
- ☐ gegen Rechnung

Meine/unsere Anschrift:

Name ....

Firma ....

Abtlg ....

Straße ....

Ort ....

Telefon ....

Datum, Unterschrift .... MS 2/89

IWT Magazin Verlags GmbH, Wendelsteinstr. 3, 8011 Vaterstetten, Tel.: 08106/3 20 57

IWT Magazin Verlags GmbH, Postfach 1144, 8011 Vaterstetten, Tel.: 08106/3 20 57

Auslieferung Schweiz: Thali AG, Industriestraße 2, CH-6285 Hitzkirch, Tel. (041) 85 28 28
Auslieferung Österreich: Erb-Verlag Ges. m.b.H. + Co. KG., Amerlingstraße 1, A-1061 Wien 6, Tel. (02 22) 587 05 26, Telex 136 145